Tkinter

GUI Programming with Python

0.1 Using current releases

The current release of python 3 is 3.5.2. The examples in this book were all run using version 3.5.0. Prior knowledge of python 3 is a requirement. Although full example python programs are included this book does not teach python 3. The examples are included to ease your learning how to make python 3 work with the Tcl/Tk widgets. Thus the programming examples are often not complete, nor expected to be complete.

Program example are displayed in a different font, Lucinda Console, and in bold. They are also indented for your convenience separating the program examples from the text. Because small snippets of code can confuse the reader I often include the full python.

The current release of Tcl/Tk is 8.6. The example programs were executed using an instance of that release.

Both python and Tcl/Tk frequently have new releases, so this book will become quickly outdated.

0.2 This book is an introduction, not a user manual.

The intent of this book is to help experienced python 3 programmers who are also novice Tcl/Tk programmers to begin to use Tcl/Tk widgets to quickly build GUI platforms. For further information, concerning Tcl/Tk the reader can take advantage of the numerous Websites on python 3 GUI development and the Website for Tcl/Tk.

This book is organized in the following sequence. Chapter one introduces the reader to the concept that both python and Tcl/Tk are interpreted language supported by their own interpreters and Websites. Chapter one introduces the first program which uses a single button. Pressing that button displays a message on the IDLE console.

Later chapters introduce Tcl/Tk widgets in a sequence that encourages the reader to develop the structure of the GUI first. Later chapters emphasize specialized techniques such as styling and event handling.

Tkinter

GUI Programming with Python

1.1 History of Tkinter

While a professor of computer science at the University of California, Berkeley John Ousterhout created the scripting language Tool Command Language (Tcl), commonly pronounced, "Tickle," for quick prototyping. The combination of Tcl with a platform independent GUI tool kit became known as Tcl/TK. Tcl gained acceptance for its comparative ease of use. Since its first release in 1988 Tcl/TK has been included in numerous computer languages. The Tcl organization maintains a list of Tcl implementations at http://wiki.tcl.tk/17975 Thus your knowledge of Tcl/TK may benefit you beyond your work in Python.

Steen Lumholt and Guido van Rossum wrote Tkinter. Basically, Tkinter is a C language program that does a straight forward simple translation from your Tkinter command to the corresponding Tcl command. Then that command is run in a Tcl interpreter. Obviously prior knowledge and experience using Tcl/Tk would clearly make your study easier. The documentation for completing advanced techniques in Tkinter often refer the reader to the Tcl documentation. The Python Software Foundation has a fine reference for tkinter at it Website at https://docs.python.org/3/library/tkinter.html#module-tkinter. In section 25.1.4 that documentation maps basic Tk into Tkinter. A brief example is included here.

Class commands in Tk correspond to class constructors in Tkinter.

button .fred =====> *fred = Button()*

This example should convince the reader of the importance of learning at least the basics of Tcl.

Because of the complexity and effort required to learn Tcl/TK many students opt to simply learn Tkinter. Most students of python programming never encounter projects requiring extensive knowledge of Tkinter and Tcl/TK. In those situations, students do not need to learn Tcl/TK.

The Tkinter module is a python wrapper built around a Tcl interpreter. Because Tkinter is included in most distributions of Python most students learn Tkinter while creating their first Python GUI program. Beginning with a GUI module containing only a single button and displaying a simple message such as "Hello world." Tkinter has been the package for implementing Python GUI programs. As a result, Tkinter has mistakenly assumed the importance of being the default module for implementing any GUI. Only after using Tkinter extensively do students ever learn about the other numerous and popular packages for creating Python GUI modules. Wikipedia maintains a list of widget tool kits at https://en.wikipedia.org/wiki/Tkinter. That list includes PyGObject, PyGTK, Pyjamas, PyQt, PySide, wxPython, and **Tkinter. Each of those tool kits has the full support of a community of loyal users who understand the benefits of its own tool kit.**

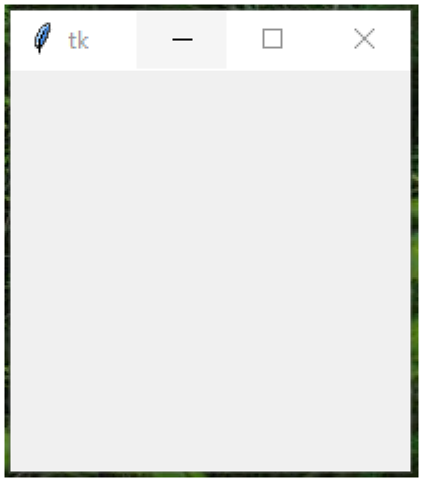
In the past Tcl/Tk has appeared to be the simpler and more rudimentary module for producing a Python GUI module. Because of the success of competing Python GUI widget tool kits Tkinter has been viewed as appropriate only for beginning students of Python GUI programming. Enhancements to the tool kit have not removed this onus. I hope this book will encourage you to reconsider Tcl/TK for your python GUI programming projects.

First Example

Our goal is to create the shortest possible GUI programming using python and Tkinter. The first line imports the Tkinter package. Next the program creates an object using the tkinter class. That object will be the main window, which is usually referred to as the root or main. Function mainloop starts the execution of our GUI program. This execution ends when you close the window using the close box. Pressing the close box creates a destroy event which calls the destroy method of the tkinter class and closes the window.

```
import tkinter as Tk

root = tk.Tk()
root.mainloop()
```

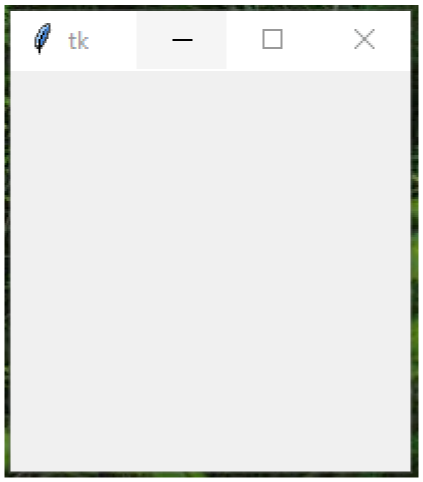
The program creates a basic and empty window with the default title of tk.



Let's rewrite this program using a class.

```
import tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

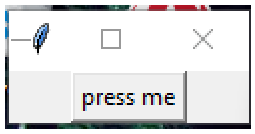
Notice the root is passed to the instance creator as the sole argument. Then every reference to a button or text box will properly reference the button in the root window. This program produces the same window.



The next GUI application will have a single button the generates a message on the python console.

```
import tkinter as tk

root = tk.Tk()
button = tk.Button(root, text="press me")
button.pack()
root.mainloop()
```

This program produces this window with a button labeled “press me.”



That button is not yet connected to any python function so pressing it does nothing. A few tkinter widgets have the command parameter which is set to the name of the function to be called. Notice the expanded python GUI program below. The button construction includes a parameter to set the callback function. The normal parentheses after a function are not included, only the name is included.

Including parentheses following he callback function's name would cause it to be executed immediately, long before the button was pushed. Also the command option was set up for Unix operating systems, not MS Windows, so the button is connected to the left click on the mouse and the space bar not the enter key as would be expected for a MS Windows OS machine. A good practice is to always use the left click on the mouse.

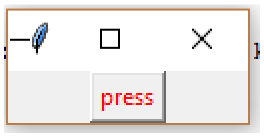
```
import tkinter as tk

def callback():
    print("click")

root = tk.Tk()
button = tk.Button(root, text="press me", command=callback)
button.pack()
root.mainloop()
```

Now the program displays “click” on the console every time you press the button.

The next program uses a class to implement the same steps.

```
import tkinter as tk

class Application(tk.Frame):
    def callback(self):
        print("click")
    def create_widgets(self):
        self.btn = tk.Button(self, text="press", fg="red", command=self.callback)
        self.btn.pack()

    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.create_widgets()

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

The new version produces the following window. Notice the foreground color of the text is red.



Pushing the button labeled red causes the program to display click on the console, exactly the same as the results of the previous version of this python program. The button constructor has a parameter named text that is set to "press." That text appears on the button as its label. The other parameter named "fg", which is an abbreviation of "foreground color", sets that text's color to red.

Naturally you want to know all the possible parameters, not simply text, fg, and command. Because tkinter makes such a direct translation of its commands to Tcl commands, the python documentation is sparse. Only a hint is provided at https://docs.python.org/3.0/library/tkinter.html?highlight=tkinter%20button#how-tk-and-tkinter-are-related. The user desiring a complete list of parameters is directed to the Tcl documentation for a button. In the section titled "A (Very) Quick Look at Tcl/Tk" the Tcl command to construct a button is

```
button  .btn  -fg red -text "press"
```

where "button" is the class name,
".btn", which begins with a dot, is the name of that button
the dash indicates that the name is the name of a constructor parameter
so the parameter is fg followed by its value of red
then the parameter text is followed by its value of press.

That Tcl command is remarkably similar to the tkinter command to create a button.

Another approach to setting the constructor options is to first create the button and later set the option values. Examine the following python GUI program.

```
from tkinter import *

class Application(Frame):
    def callback(self):
        print("click")

    def createWidgets(self):
        self.btn = Button(self)
        self.btn["text"] = "press",
        self.btn["fg"] = "red",
        self.btn["command"] = self.callback
        self.btn.pack()

    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.createWidgets()

root = Tk()
app = Application(master=root)
app.mainloop()
root.destroy()
```

The button constructor returns a dictionary. The constructor option is the key and the option's value becomes the the value of the key value pair in the dictionary. This program produces the same red button labeled “press” and displays “click” on the console. Also notice the final command which destroys the window thus cleaning up.

The next program prints the key value pairs, so we can see the list of constructor options for a button.

```
import tkinter as tk

root = tk.Tk()

button = tk.Button(root, text="press me")

d = button.config()

for k in d:

    print(k, d[k])

root.mainloop()
```

The results are listed below.

```
width ('width', 'width', 'Width', 0, 0)
compound ('compound', 'compound', 'Compound', <index object: 'none'>, 'none')
repeatinterval ('repeatinterval', 'repeatInterval', 'RepeatInterval', 0, 0)
relief ('relief', 'relief', 'Relief', <index object: 'raised'>, 'raised')
height ('height', 'height', 'Height', 0, 0)
fg ('fg', '-foreground')
wraplength ('wraplength', 'wrapLength', 'WrapLength', <pixel object: '0'>, <pixel object: '0'>)
padx ('padx', 'padX', 'Pad', <pixel object: '1'>, <pixel object: '1'>)
repeatdelay ('repeatdelay', 'repeatDelay', 'RepeatDelay', 0, 0)
state ('state', 'state', 'State', <index object: 'normal'>, 'normal')
activebackground ('activebackground', 'activeBackground', 'Foreground', <border object:
'SystemButtonFace'>, 'SystemButtonFace')
text ('text', 'text', 'Text', '', 'press me')
background ('background', 'background', 'Background', <border object: 'SystemButtonFace'>,
'SystemButtonFace')
image ('image', 'image', 'Image', '', '')
default ('default', 'default', 'Default', <index object: 'disabled'>, 'disabled')
highlightcolor ('highlightcolor', 'highlightColor', 'HighlightColor', <color object:
'SystemWindowFrame'>, 'SystemWindowFrame')
underline ('underline', 'underline', 'Underline', -1, -1)
activeforeground ('activeforeground', 'activeForeground', 'Background', <color object:
'SystemButtonText'>, 'SystemButtonText')
highlightbackground ('highlightbackground', 'highlightBackground', 'HighlightBackground',
<border object: 'SystemButtonFace'>, 'SystemButtonFace')
bg ('bg', '-background')
disabledforeground ('disabledforeground', 'disabledForeground', 'DisabledForeground', <color
object: 'SystemDisabledText'>, 'SystemDisabledText')
foreground ('foreground', 'foreground', 'Foreground', <color object: 'SystemButtonText'>,
'SystemButtonText')
pady ('pady', 'padY', 'Pad', <pixel object: '1'>, <pixel object: '1'>)
bitmap ('bitmap', 'bitmap', 'Bitmap', '', '')
overrelief ('overrelief', 'overRelief', 'OverRelief', '', '')
textvariable ('textvariable', 'textVariable', 'Variable', '', '')
font ('font', 'font', 'Font', <font object: 'TkDefaultFont'>, 'TkDefaultFont')
```

```
cursor ('cursor', 'cursor', 'Cursor', '', '')
borderwidth ('borderwidth', 'borderWidth', 'BorderWidth', <pixel object: '2'>, <pixel object: '2'>)
anchor ('anchor', 'anchor', 'Anchor', <index object: 'center'>, 'center')
takefocus ('takefocus', 'takeFocus', 'TakeFocus', '', '')
justify ('justify', 'justify', 'Justify', <index object: 'center'>, 'center')
highlightthickness ('highlightthickness', 'highlightThickness', 'HighlightThickness', <pixel
object: '1'>, <pixel object: '1'>)
command ('command', 'command', 'Command', '', '')
bd ('bd', '-borderwidth')
```

Alternatively you can use the config() method to update multiple attributes after the button object was created.

```
btn.config(fg=”red”, text=”press”)
```

The Tcl manual at https://www.tcl.tk/man/tcl8.4/TkCmd/button.htm displays a list of the button constructors options.

button - Create and manipulate button widgets

SYNOPSIS

button *pathName* ?*options*?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify

-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-repeatdelay, repeatDelay, RepeatDelay
-repeatinterval, repeatInterval, RepeatInterval
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

WIDGET-SPECIFIC OPTIONS

-command, command, Command
-default, default, Default
-height, height, Height
-overrelief, overRelief, OverRelief
-state, state, State
-width, width, Width

DESCRIPTION
WIDGET COMMAND

pathName **cget** *option*
pathName **configure** ?*option*? ?*value option value* ...?
pathName **flash**
pathName **invoke**

And New Mexico Tech published the copyrighted python tkinter equivalent table at

http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/button.html.

Notice how similar the Tcl attribute names are to the tkinter contructor attribute names. Many programmers simply use the Tcl documentation or print the options themselves. Understanding the impact of each attribute's value demands either extensive experience using tkinter or Tcl.

Widgets

Tkinter Widgets

2.1 Overview of Widgets

Tcl/Tk provides both a standard tool kit of widgets known as tk and a themed tool kit known as ttk. The themed widgets assume the default appearance of the operating system, so ttk widgets should be the future of Python's Tkinter package. Thus this book will switch from using tk to ttk widgets.

In addition to eleven widgets provided by tk which include Button, CheckButton, Entry, Frame, Label, LabelFrame, MenuButton, PanedWindow, Radiobutton, Scale and Scrollbar ttk provided Combobox, Notebook, Progressbar, Separator, Sizegrip and Treeview. Each of those classes is a subclass of Widget.

2.2 ttk Button

Let's rewrite the example program from chapter 1 using the ttk Button, instead of a tk button. We import the ttk widgets and change any occurrence of tk to ttk. Notice the foreground color for the button is no longer set. Ttk widgets use a style object to set visual properties. Thus tk commands are different from ttk commands. We will discuss the required style commands later.

```
import tkinter as tk
from tkinter import *
import tkinter.ttk as ttk
from tkinter.ttk import *

class Application(tk.Frame):
    def callback(self):
        print("click")

    def create_widgets(self):
        self.btn = ttk.Button(self, text="press",
               command=self.callback)
        self.btn.pack()

    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.create_widgets()

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

This python application was run on a PC with Windows 10, so the button's appearance matches the theme for Windows 10. That illustrates the meaning of themed tool kit widgets. This button would have a different appearance if the application ran on a Linux box, or Apple computer. The button would always conform to the design theme of the OS it is run on.

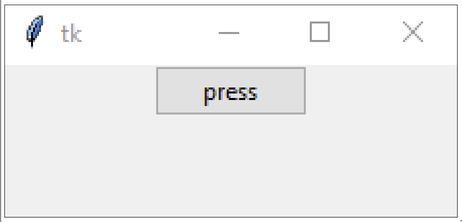


Illustration 1: unselected button

With the cursor hovering over the button, it changes its appearance according to the Windows 10 theme.

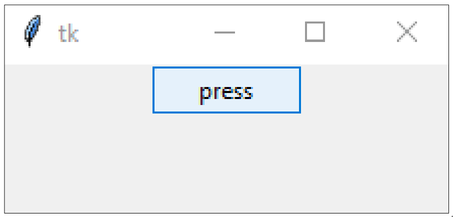


Illustration 2: selected button

2.3 ttk Label

Next we enhance our program to echo your name. We begin by including a label. Notice the new pack command following the command to create the label. All GUI items must be packed before the program is executed.

```
import tkinter as tk
from tkinter import *
import tkinter.ttk as ttk
from tkinter.ttk import *

class Application(tk.Frame):
    def callback(self):
        print("click")

    def create_widgets(self):
        self.lbl = ttk.Label(self, text="First Name")
        self.lbl.pack()
        self.btn = ttk.Button(self, text="press",
                 command=self.callback)
        self.btn.pack()
    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.create_widgets()

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

This new label simply displays "First Name."

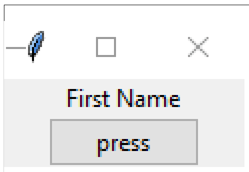


Illustration 3: side=LEFT

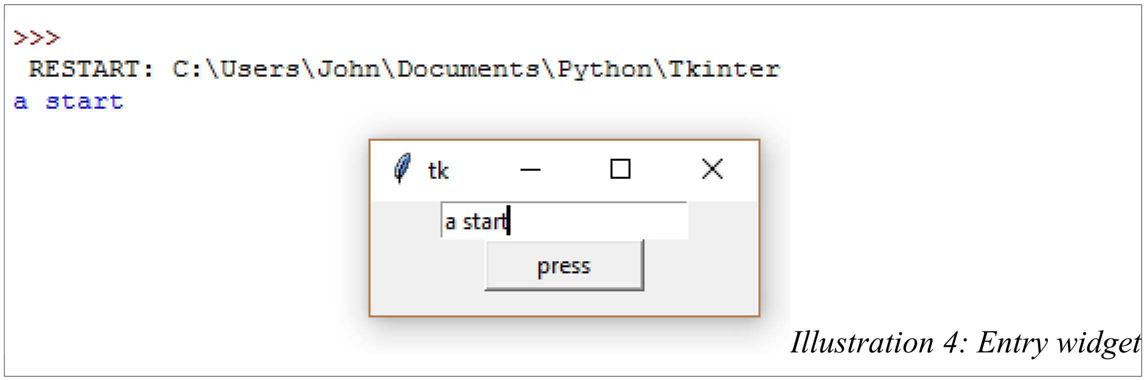
Recall the ttk button is similar to the tk button except that the ttk button's appearance matches the underlying operating system's theme. Likewise the ttk label is similar to the tk label. The ttk widgets are the preferred widgets, so there is little benefit to gain from discussing any tk label widgets when we could discuss the corresponding ttk widget.

2.4 ttk Entry

Next we enhance our program to echo your name. We begin by including an entry widget.

```
from tkinter import *

master = Tk()

e = Entry(master)
e.pack()

e.focus_set()

def callback():
    print(e.get())

b = Button(master, text="get", width=10, command=callback)
b.pack()

mainloop()
```

Running this program produced the following screen into which as enter "a start." Notice the get function returns the text contents of the entry. The program prints that text on the console. Also the focus_set function places the cursor at the start of that entry.



```
>>>
 RESTART: C:\Users\John\Documents\Python\Tkinter
a start
```

Illustration 4: Entry widget

2.5 ttk Frame

Next let's change our python program to use an object-oriented approach. Our next example program uses the ttk frame, which is an object for holding other widgets, themselves objects. Simply generating a frame will satisfy our goal of thinking in terms of objects.

```
import tkinter
from tkinter import ttk

class Application(ttk.Frame):
    def __init__(self, parent, *args, **kwargs):
        ttk.Frame.__init__(self, parent, *args, **kwargs)
        self.root = parent

if __name__ == '__main__':
    root = tkinter.Tk()
    Application(root)
    root.mainloop()
```

Running this program produce an empty window, which is described in both TCL and tkinter as a frame. The frame appears normal to users of both MS windows and unix systems. The title bar includes an icon representing the application, window title, and minimizer, maximizer, and close icons.

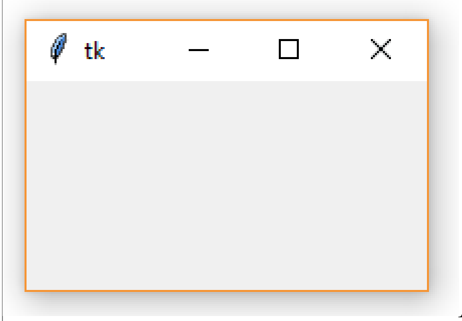


Illustration 5: ttk.Frame

The object named root is returned by the frame constructor so root is a frame. The class application expects an argument of type frame and applications will accept root as an object of type frame.

2.6 pack()

Notice the pack command contains arguments. Thus we must learn the pack manager which arranges widgets within their parent widget. A frame is a widget with a default size and location.

You can also use the ipadx and ipady options for internal packing surrounding the contents. For example this python program ...

```
import tkinter as tk
from tkinter import *
import tkinter.ttk as ttk
from tkinter.ttk import *

class Application(tk.Frame):
    def callback(self):
        print("click")

    def create_widgets(self):
        self.lbl = ttk.Label(self, text="First Name")
        self.lbl.pack(ipadx = 2, ipady = 3, padx = 4, pady = 5)
        self.btn = ttk.Button(self, text="press",
                              command=self.callback)
        self.btn.pack(ipadx = 6, ipady = 7, padx = 8, pady = 10)

    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.create_widgets()

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

produces this result.

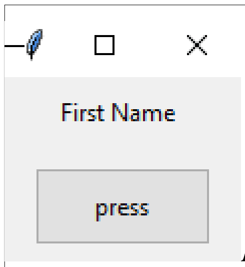


Illustration 6: padding

Notice both the padding surrounding the label and button and the internal padding surrounding the text for each object.

The pack command attempts to put each widget within its parent widget. For this example root is the parent of both the label and the button. Pack automatically puts the widget in the center. Additional widgets are centered below. When all widgets are included the parent widget is automatically sized to fit all the internal widgets. Thus the window shown above is exactly large enough to hold both the label and button. Of course the width of the parent equals the width of the widest child widget and its padding, so windows are only as big as needed to hold all the widgets.

When the parameter of side = LEFT is added then the widget are moved to the left edge of its parent. Repeatedly using side = LEFT will align the widgets in a horizontal row. See the program below.

```
import tkinter as tk
from tkinter import *
import tkinter.ttk as ttk
from tkinter.ttk import *

class Application(tk.Frame):
    def callback(self):
        print("click")

    def create_widgets(self):
        self.lbl = ttk.Label(self, text="First Name")
        self.lbl.pack(ipadx = 2, ipady = 3, padx = 4,
          pady = 5, side = LEFT)
        self.btn = ttk.Button(self, text="press",
                               command=self.callback)
        self.btn.pack(ipadx = 6, ipady = 7, padx = 8,
          pady = 10, side = LEFT)

    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.create_widgets()

root = tk.Tk()
app = Application(master=root)
```

```
    app.mainloop()
```

That program produces this result.

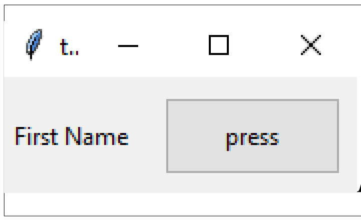


Illustration 7: side=LEFT

The program below used side = RIGHT.

```
    import tkinter as tk
    from tkinter import *
    import tkinter.ttk as ttk
    from tkinter.ttk import *

    class Application(tk.Frame):
        def callback(self):
            print("click")

        def create_widgets(self):
            self.lbl = ttk.Label(self, text="First Name")
            self.lbl.pack(ipadx = 2, ipady = 3, padx = 4,
              pady = 5, side = RIGHT)
            self.btn = ttk.Button(self, text="press",
                                  command=self.callback)
            self.btn.pack(ipadx = 6, ipady = 7, padx = 8,
              pady = 10, side = RIGHT)

        def __init__(self, master=None):
            super().__init__(master)
            self.pack()
            self.create_widgets()

    root = tk.Tk()
    app = Application(master=root)
    app.mainloop()
```

It produced a different result.

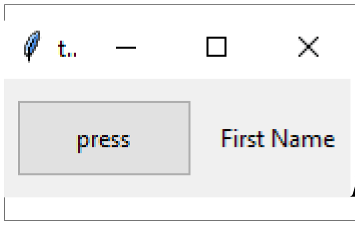


Illustration 8: side=RIGHT

Notice the label is at the right.

Many applications depend upon a status bar at the bottom of the window. For tkinter applications the programmer should construct a label and place it at the bottom of the frame.

```
status = Label(master, text="normal", anchor=W)
status.pack(side=BOTTOM, fill=X)
```

The entire python program is shown below.

```
import tkinter as tk
from tkinter import *
import tkinter.ttk as ttk
from tkinter.ttk import *

class Application(tk.Frame):
    def callback(self):
        #print("click")
        self.status.config(text = "Button pressed")

    def create_widgets(self):
        self.btn = ttk.Button(self, text="press",
                              command=self.callback)
        self.btn.pack(ipadx = 6, ipady = 7, padx = 8,
     pady = 10, side = RIGHT)
        self.status = Label(self, text="status: normal")
        self.status.pack(side=BOTTOM, fill=X)

    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.create_widgets()

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

Running that python program produces this GUI.

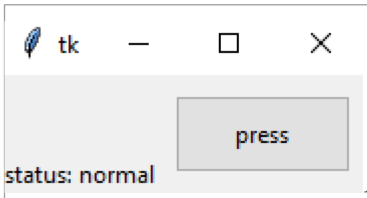
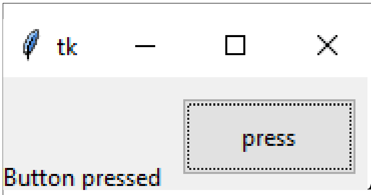


Illustration 9: status bar

Pressing the button changes the status line to “button pressed.”



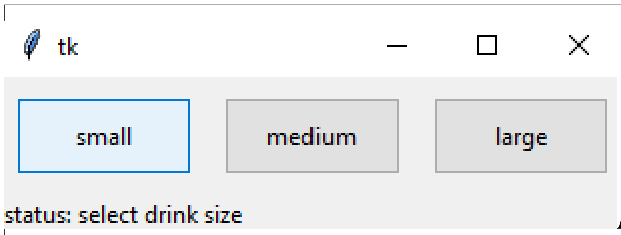
Button pressed

Illustration 10: updated status bar

Not all applications have a status bar. Those few applications who benefit from having a status bar always display that status bar at the bottom of the window. Changing its location might confuse users who have become accustomed to any status bar being displayed at the bottom of the window.

2.7 lambda

Providing separate buttons for each choice might lead us to create separate callback functions for each button. That would enlarge our program and make the program more difficult to read and work with. Fortunately python allows the use of the lambda option in the command parameter of a button. As shown below this problem involves three buttons to select a drink's size. Each button has a command parameter of command=lambda: self.callback(“small”) or “medium” or “large.” The callback function simply adds that word to the status message at the bottom of the window.



status: select drink size

Illustration 11: choice of three buttons

Each button has a command parameter of command=lambda: self.callback(“small”) or “medium” or “large.”
The callback function simply adds that word to the status message at the bottom of the window.

The entire python program is shown here.

```
import tkinter as tk
from tkinter import *
import tkinter.ttk as ttk
from tkinter.ttk import *

class Application(tk.Frame):
    def callback(self, size):
        msg = "status: selected " + size
        self.status.config(text = msg)
    def create_widgets(self):
        self.status = Label(self, text="status: select drink
size")
        self.status.pack(side = BOTTOM, fill=X)
        self.small_btn = ttk.Button(self, text="small",
command=lambda: self.callback("small"))
        self.small_btn.pack(ipadx = 6, ipady = 7, padx = 8,
pady = 10, side = LEFT)
        self.medium_btn = ttk.Button(self, text="medium",
command=lambda: self.callback("medium"))
        self.medium_btn.pack(ipadx = 6, ipady = 7, padx = 8,
          pady = 10, side = LEFT)
        self.large_btn = ttk.Button(self, text="large",
command=lambda: self.callback("large"))
        self.large_btn.pack(ipadx = 6, ipady = 7, padx = 8,
          pady = 10, side = LEFT)

    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.create_widgets()

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

The lambda callback function is limited to one parameter.

ttk Widgets

More Tkinter ttk Widgets

3.1 Overview

This chapter introduces additional ttk widgets. Beyond expanding our awareness of the possible Tk and ttk widgets, this chapter also introduces the grid layout system. Recall that tk provides eleven widgets: Button, CheckButton, Entry, Frame, Label, LabelFrame, MenuButton, PanedWindow, Radiobutton, Scale and Scrollbar. Ttk provides Combobox, Notebook, Progressbar, Separator, Sizegrip and Treeview. Such a huge list exceeds one chapter, so this chapter will selectively introduce a few more ttk widgets.

3.2 Life Expectancy

Using open source data provided by the Center for Disease Control it is possible to develop a python program to calculate a person's life expectancy. The Social Security Administration has already extracted the tables of information from data provided by the CDC. The Social Security Administration's table are located on the Internet at Source https://www.ssa.gov/oact/STATS/. Table table4c6.html provides the expected remaining years when only the person's age and gender is provided.

Portion of ..

Period Life Table, 2013

Exact age	*Male*			*Female*
	Death probability [a]	*Number of lives* [b]	*Life expectancy*	*Death probability* [a]
0	*0.006519*	*100,000*	*76.28*	*0.005377*
1	*0.000462*	*99,348*	*75.78*	*0.000379*

The Period Life Table of 2013 uses a person's age as the key. That age is the whole number of years fully completed. No fractions are allowed as the key. For that table age can range from zero to 119. Both the male and female life expectancy are the projected number of remaining years.

Using that table's data a python program has a dictionary with key of age and gender and value of the remaining years. The first few lines are displayed here.

```
yrs = {(0,'m') : 76.28,
        (0,'f') : 81.05,
        (1,'m') : 75.78,
        (1,'f') : 80.49,
```

Next the longevity program must ask the user's age and gender. The two easiest widgets to use are ttk Scale or ttk Combobox.

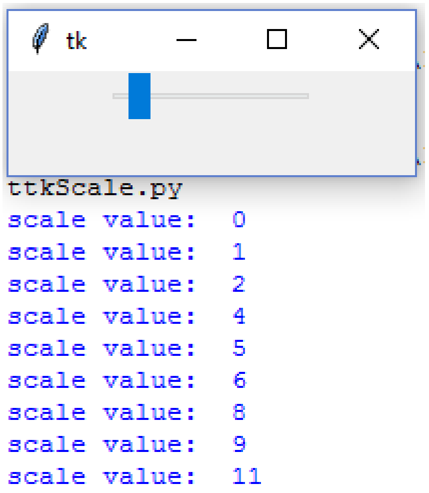
3.3 ttk.Scale

The ttk.Scale widget provides an unlabeled slider. Thus the programmer might add a label or button. Using ttk widgets would allow matching the style of the operating system. The programmer has the responsibility of updating the value displayed on that label.

You can set the orientation as horizontal or vertical. The range can also be set.

```
import tkinter
from tkinter import ttk

class Application(ttk.Frame):
    def __init__(self, parent, *args, **kwargs):
        ttk.Frame.__init__(self, parent, *args, **kwargs)
        self.root.title("life expectancy")

    def __init__(self, master):
        c = ttk.Scale(master, from_=0, to=120,
                      orient=tkinter.HORIZONTAL,
                      command=self.scale_update)
        c.pack()

    def scale_update(self, evt):
        idx = int(float(evt))
        print("scale value: ", idx)

if __name__ == '__main__':
    root = tkinter.Tk()
    Application(root)
    root.mainloop()
```

This program produces the following window with a scale.



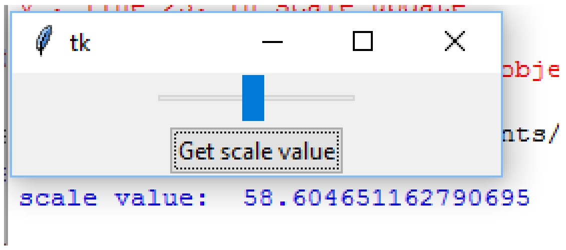
```
ttkScale.py
scale value:  0
scale value:  1
scale value:  2
scale value:  4
scale value:  5
scale value:  6
scale value:  8
scale value:  9
scale value:  11
```

Notice that the program has displayed the scale's value on the console as the scale was moved from zero to eleven. The scale provides a float for the current position. Thus the program used idx = int(float(evt)) to get the scale's current position as an integer value. Obviously using the function round would provide an more accurate value than simply using the int function to truncate the float value.

This example program uses the var option with a DoubleVar to hold the scale's value. Then pressing a button calls a function to get the value from that DoubleVar.

```
import tkinter
from tkinter import *
from tkinter import ttk

class Application(ttk.Frame):
    global v
    def __init__(self, parent, *args, **kwargs):
        ttk.Frame.__init__(self, parent, *args, **kwargs)
        self.root.title("life expectancy")

    def __init__(self, master):
        self.v = tkinter.DoubleVar()
        c = ttk.Scale(master, from_=0, to=120,
                   orient=tkinter.HORIZONTAL,
                   var = self.v)
        c.pack()
        b = ttk.Button(master, text="Get scale value",
                   command=self.scale_update)
        b.pack()

    def scale_update(self):
        value = self.v.get()
        print("scale value: ", value)

if __name__ == '__main__':
    root = tkinter.Tk()
    Application(root)
    root.mainloop()
```

Notice DoubleVar has been provided by the tkinter package. This program produced this window



```
scale value:  58.604651162790695
```

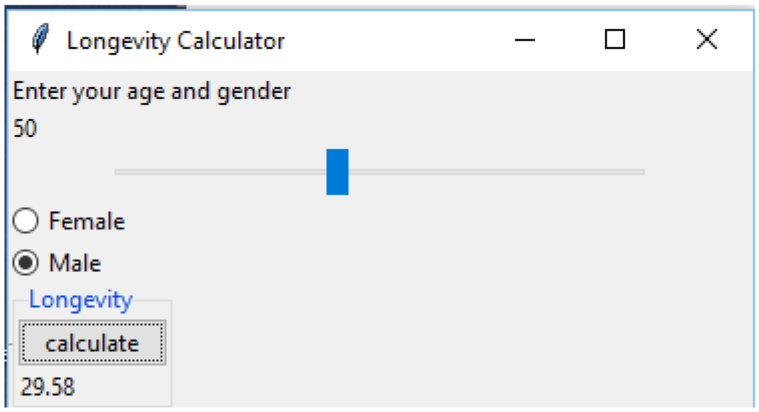
The value parameter can be used to set the initial value for the scale, which defaults to a float of zero.

The longevity program shown below uses the first method to set the value of a label. Thus the user can easily see the impact of moving the slider on the scale.

```
    import tkinter
    from tkinter import ttk

    # See http://www.datadependence.com/2016/04/how-to-build-gui-in-python-3/ for pack
    directions
    class App(ttk.Frame):
        """Longevity calculator"""
        def __init__(self, parent, *args, **kwargs):
            ttk.Frame.__init__(self, parent, *args, **kwargs)
            self.root = parent
            self.init_gui(parent)

        def calc_life_expectancy(self):
            # Source https://www.ssa.gov/oact/STATS/table4c6.html
            # gender female:1 male:2
            years_old = int(self.age_lbl['text'])
            if self.gender.get() == 1:
                sex = "f"
            else:
                sex = "m"
            yrs = {(0,'m') : 76.28,
                   (0,'f') : 81.05,
                   (1,'m') : 75.78,
                   (1,'f') : 80.49,
                   (2,'m') : 74.82,
                   (2,'f') : 79.52,
                   (3,'m') : 73.84,
                   (3,'f') : 78.54,
                   (4,'m') : 72.85,
                   (4,'f') : 77.55,
                   (5,'m') : 71.87,
                   (5,'f') : 76.56,
                   (6,'m') : 70.88,
                   (6,'f') : 75.57,
                   (7,'m') : 69.89,
                   (7,'f') : 74.58,
                   (8,'m') : 68.90,
                   (8,'f') : 73.58,
                   (9,'m') : 67.90,
                   (9,'f') : 72.59,
                   (10,'m') : 66.91,
                   (10,'f') : 71.60,
                   (11,'m') : 65.92,
                   (11,'f') : 70.60,
                   (12,'m') : 64.92,
                   (12,'f') : 69.61,
                   (13,'m') : 63.93,
                   (13,'f') : 68.62,
                   (14,'m') : 62.94,
                   (14,'f') : 67.63,
                   (15,'m') : 61.96,
                   (15,'f') : 66.64,
                   (16,'m') : 60.99,
                   (16,'f') : 65.65,
                   (17,'m') : 60.02,
                   (17,'f') : 64.67,
                   (18,'m') : 59.05,
                   (18,'f') : 63.68,
                   (19,'m') : 58.09,
                   (19,'f') : 62.70,
                   (20,'m') : 57.14,
                   (20,'f') : 61.72,
                   (21,'m') : 56.20,
                   (21,'f') : 60.75,
                   (22,'m') : 55.27,
                   (22,'f') : 59.77,
                   (23,'m') : 54.33,
                   (23,'f') : 58.80,
                   (24,'m') : 53.40,
                   (24,'f') : 57.82,
                   (25,'m') : 52.47,
                   (25,'f') : 56.85,
                   (26,'m') : 51.54,
                   (26,'f') : 55.88,
                   (27,'m') : 50.61,
                   (27,'f') : 54.91,
                   (28,'m') : 49.68,
                   (28,'f') : 53.94,
                   (29,'m') : 48.75,
                   (29,'f') : 52.97,
                   (30,'m') : 47.82,
                   (30,'f') : 52.01,
                   (31,'m') : 46.89,
                   (31,'f') : 51.04,
                   (32,'m') : 45.96,
                   (32,'f') : 50.08,
                   (33,'m') : 45.03,
                   (33,'f') : 49.11,
                   (34,'m') : 44.10,
                   (34,'f') : 48.15,
                   (35,'m') : 43.17,
                   (35,'f') : 47.19,
                   (36,'m') : 42.24,
                   (36,'f') : 46.23,
                   (37,'m') : 41.31,
                   (37,'f') : 45.28,
                   (38,'m') : 40.38,
                   (38,'f') : 44.33,
                   (39,'m') : 39.46,
                   (39,'f') : 43.37,
                   (40,'m') : 38.53,
                   (40,'f') : 42.43,
                   (41,'m') : 37.61,
                   (41,'f') : 41.48,
                   (42,'m') : 36.70,
                   (42,'f') : 40.54,
                   (43,'m') : 35.78,
                   (43,'f') : 39.60,
                   (44,'m') : 34.88,
                   (44,'f') : 38.66,
                   (45,'m') : 33.98,
                   (45,'f') : 37.73,
                   (46,'m') : 33.08,
                   (46,'f') : 36.81,
                   (47,'m') : 32.19,
                   (47,'f') : 35.89,
                   (48,'m') : 31.32,
                   (48,'f') : 34.97,
                   (49,'m') : 30.44,
                   (49,'f') : 34.06,
                   (50,'m') : 29.58,
                   (50,'f') : 33.16,
                   (51,'m') : 28.73,
                   (51,'f') : 32.27,
                   (52,'m') : 27.89,
                   (52,'f') : 31.38,
                   (53,'m') : 27.05,
                   (53,'f') : 30.49,
                   (54,'m') : 26.23,
                   (54,'f') : 29.62,
                   (55,'m') : 25.41,
                   (55,'f') : 28.74,
                   (56,'m') : 24.61,
                   (56,'f') : 27.88,
                   (57,'m') : 23.82,
                   (57,'f') : 27.01,
                   (58,'m') : 23.03,
                   (58,'f') : 26.16,
                   (59,'m') : 22.25,
                   (59,'f') : 25.31,
                   (60,'m') : 21.48,
                   (60,'f') : 24.46,
                   (61,'m') : 20.72,
                   (61,'f') : 23.62,
                   (62,'m') : 19.97,
                   (62,'f') : 22.78,
                   (63,'m') : 19.22,
                   (63,'f') : 21.95,
                   (64,'m') : 18.48,
                   (64,'f') : 21.13,
                   (65,'m') : 17.75,
                   (65,'f') : 20.32,
                   (66,'m') : 17.03,
                   (66,'f') : 19.52,
                   (67,'m') : 16.32,
                   (67,'f') : 18.73,
                   (68,'m') : 15.61,
                   (68,'f') : 17.95,
                   (69,'m') : 14.92,
                   (69,'f') : 17.18,
                   (70,'m') : 14.24,
                   (70,'f') : 16.43,
                   (71,'m') : 13.57,
                   (71,'f') : 15.68,
                   (72,'m') : 12.92,
                   (72,'f') : 14.95,
                   (73,'m') : 12.27,
                   (73,'f') : 14.23,
                   (74,'m') : 11.65,
                   (74,'f') : 13.53,
                   (75,'m') : 11.03,
                   (75,'f') : 12.83,
                   (76,'m') : 10.43,
                   (76,'f') : 12.16,
                   (77,'m') : 9.85,
                   (77,'f') : 11.50,
                   (78,'m') : 9.28,
                   (78,'f') : 10.86,
                   (79,'m') : 8.73,
                   (79,'f') : 10.24,
                   (80,'m') : 8.20,
                   (80,'f') : 9.64,
                   (81,'m') : 7.68,
                   (81,'f') : 9.05,
                   (82,'m') : 7.19,
                   (82,'f') : 8.48,
                   (83,'m') : 6.72,
                   (83,'f') : 7.94,
                   (84,'m') : 6.27,
                   (84,'f') : 7.42,
                   (85,'m') : 5.84,
                   (85,'f') : 6.92,
                   (86,'m') : 5.43,
                   (86,'f') : 6.44,
                   (87,'m') : 5.04,
                   (87,'f') : 5.99,
                   (88,'m') : 4.68,
                   (88,'f') : 5.57,
                   (89,'m') : 4.34,
                   (89,'f') : 5.17,
                   (90,'m') : 4.03,
                   (90,'f') : 4.80,
                   (91,'m') : 3.74,
                   (91,'f') : 4.45,
                   (92,'m') : 3.47,
                   (92,'f') : 4.13,
                   (93,'m') : 3.23,
                   (93,'f') : 3.84,
                   (94,'m') : 3.01,
                   (94,'f') : 3.57,
                   (95,'m') : 2.82,
                   (95,'f') : 3.34,
                   (96,'m') : 2.64,
                   (96,'f') : 3.12,
                   (97,'m') : 2.49,
                   (97,'f') : 2.93,
                   (98,'m') : 2.36,
                   (98,'f') : 2.76,
                               (99,'m') : 2.24,
                               (99,'f') : 2.60,
                               (100,'m') : 2.12,
                               (100,'f') : 2.45,
                               (101,'m') : 2.01,
                               (101,'f') : 2.30,
                               (102,'m') : 1.90,
                               (102,'f') : 2.17,
                               (103,'m') : 1.80,
                               (103,'f') : 2.03,
                               (104,'m') : 1.70,
                               (104,'f') : 1.91,
                               (105,'m') : 1.60,
                               (105,'f') : 1.78,
                               (106,'m') : 1.51,
                               (106,'f') : 1.67,
                               (107,'m') : 1.42,
                               (107,'f') : 1.56,
                               (108,'m') : 1.34,
                               (108,'f') : 1.45,
                               (109,'m') : 1.26,
                               (109,'f') : 1.35,
                               (110,'m') : 1.18,
                               (110,'f') : 1.26,
                               (111,'m') : 1.11,
                               (111,'f') : 1.17,
                               (112,'m') : 1.04,
                               (112,'f') : 1.08,
                               (113,'m') : 0.97,
                               (113,'f') : 1.00,
                               (114,'m') : 0.90,
                               (114,'f') : 0.92,
                               (115,'m') : 0.84,
                               (115,'f') : 0.85,
                               (116,'m') : 0.78,
                               (116,'f') : 0.78,
                               (117,'m') : 0.72,
                               (117,'f') : 0.72,
                               (118,'m') : 0.67,
                               (118,'f') : 0.67,
                               (119,'m') : 0.61,
                               (119,'f') : 0.61,
                               }
                           if (years_old,sex) in yrs:
                               self.answer_label['text'] = str(yrs.get((years_old,sex)))
                           else:
                               self.answer_label['text'] = str(81.05)

                   """
```

```
# Source: Social Security Administration, Estimates from the 2016 Trustees Report.
"""

    def dsp_age(self,e):
        self.age_lbl.config(text = str(round(int(float(e)))))

    def init_gui(self, parent):
        """Builds GUI."""
        self.root.title('Longevity Calculator')

        # Labels that remain constant throughout execution.
        ttk.Label(parent, text='Enter your age and gender').pack(anchor="w",fill = tkinter.X)
        self.age_lbl = ttk.Label(parent, text='1')
        self.age_lbl.pack(anchor="w")
        self.age = tkinter.IntVar()
        c = ttk.Scale(parent, from_=0, to=120,
                      variable = self.age,
                      orient=tkinter.HORIZONTAL,
                      length="200p",
                      command = self.dsp_age)
        c.set(45)
        c.pack()
        #ttk.Separator(parent, orient='horizontal').pack(anchor="w", fill = tkinter.X)
        self.gender = tkinter.IntVar()
        ttk.Radiobutton(parent, text="Female", variable=self.gender,
value=1).pack(anchor=tkinter.W)
        ttk.Radiobutton(parent, text="Male", variable=self.gender,
value=2).pack(anchor=tkinter.W, fill = tkinter.X)

        ttk.Separator(parent, orient='horizontal').pack(anchor="w", side=tkinter.LEFT, fill =
tkinter.X)

        self.answer_frame = ttk.LabelFrame(parent, text='Longevity',
            height=100)
        self.answer_frame.pack(anchor=tkinter.W)
        self.calc = ttk.Button(self.answer_frame, text = "calculate",
command=self.calc_life_expectancy)
        self.calc.pack(anchor=tkinter.W)
        self.answer_label = ttk.Label(self.answer_frame, text='78.8')
        self.answer_label.pack(anchor=tkinter.W)

if __name__ == '__main__':
    root = tkinter.Tk()
    App(root)
    root.mainloop()
```

This program generates the window shown below.



Notice how easily I inserted an error into my python program. The data supports years from 0 to 119, but the scale varies from 0 to 120. At the age of 120 you will erroneously have the default value of 81 years remaining.

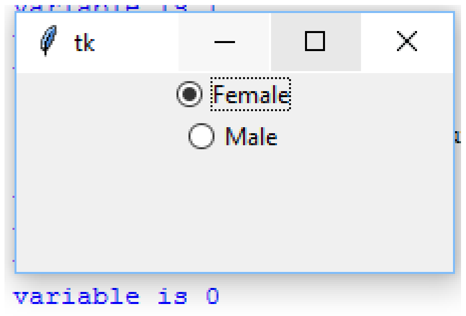
Realize that ttk widgets do not automatically work with your program's logic. You must carefully test your code. Associating a variable with a widget often avoids timing issues. The command for the first approach to using a ttk scale widget called a function that got the scale's value from a python variable which had been set to the scale. Another approach would have used both the variable and command parameters. Then the function associated with the command would get that scale's value from its associated tk variable, thus avoiding timing problems.

3.4 ttk.RadioButton

The ttk.RadioButton provides a simple method to select one option from a list. This example python program uses a gang of two RadioButtons linked by their use of the same variable.

```
import tkinter
from tkinter import ttk

class Application(ttk.Frame):
    def __init__(self, parent, *args, **kwargs):
        ttk.Frame.__init__(self, parent, *args, **kwargs)
        self.root.title("Select gender")

    def __init__(self, master):
        self.var = tkinter.IntVar()
        c = ttk.Radiobutton(
            master, text="Female",
            variable=self.var, value=1,
            command=self.cb)
        c.pack()
        d = ttk.Radiobutton(
            master, text="Male",
            variable=self.var, value=0,
            command=self.cb)
        d.pack()

    def cb(self):
        print("variable is", self.var.get())

if __name__ == '__main__':
    root = tkinter.Tk()
    Application(root)
    root.mainloop()
```

The value provided for each widget is the value displayed below on the console when a ttk.RadioButton is selected.



```
variable is 0
variable is 1
```

The widgets share the same variable, so when the user selects the Female option then all other options, which for this program is the Male option, are automatically shut off. Only one of the ganged values can be selected at a time.

3.5 ttk.LabelFrame

The ttk.LabelFrame provides a labeled frame surrounding a group of widgets. Thus the programmer can easily establish a visual grouping of widgets. Notice the ttk.LabelFrame from the longevity program.

```
        self.answer_frame = ttk.LabelFrame(parent, text='Longevity',
                height=100)
        self.answer_frame.pack(anchor=tkinter.W)
        self.calc = ttk.Button(self.answer_frame, text = "calculate",
    command=self.calc_life_expectancy)
        self.calc.pack(anchor=tkinter.W)
        self.answer_label = ttk.Label(self.answer_frame, text='78.8')
        self.answer_label.pack(anchor=tkinter.W)
```

The widgets within the LabelFrame use the variable holding the

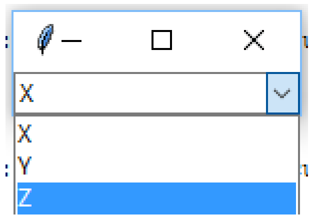
LabelFrame as the master, or parent, widget. Thus the widget is attached to the LabelFrame. Notice at the bottom of the window, shown above, how the LabelFrame surrounds the enclosed label widget and button widget.

3.5 ttk.ComboBox

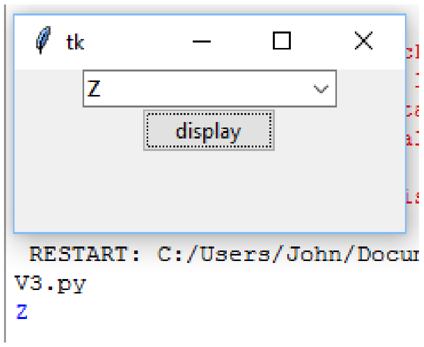
Some users find sliders difficult to use. They can become especially frustrated when their window is very small, such as on a telephone, and the range is large. Another approach is to use a combobox. The combobox provides a drop down list of options from which the user can select the appropriate option. Admittedly selecting your birth year from the range of 0 to 120 can be frustrating. The Social Security Administration, travel sites, and many longevity calculators employ such long lists for ages and years of birth. Although annoying to use they are easy to comprehend. The following example program demonstrates the usage of a ComboBox.

```
import tkinter
from tkinter import *
from tkinter import ttk as ttk
from tkinter.ttk import *

class App:

    value_of_combo = 'X'

    def __init__(self, parent):
        self.parent = parent
        self.value_of_combo = 'X'
        self.combo()

    def newselection(self):
        self.value_of_combo = self.box.get()
        print(self.value_of_combo)

    def combo(self):
        self.box_value = tkinter.StringVar()
        self.box = ttk.Combobox(self.parent, textvariable=self.box_value)
        self.box['values'] = ('X', 'Y', 'Z')
        self.box.current(0)
        self.box.pack()
        self.btn = ttk.Button(self.parent, text="display", command=self.newselection)
        self.btn.pack()

if __name__ == '__main__':
    root = tkinter.Tk()
    app = App(root)
    root.mainloop()
```

This program produces the following window providing options of X, Y, and Z with the default option of X already in the box.



The user is in the process of selecting option Z, which will be placed in the box and displayed on the console.



```
 RESTART: C:/Users/John/Docu
V3.py
Z
```

Warning: the user can type any value into the value box, instead of selecting from the list provided by the combobox, so your python program should check the validity of that value.

3.6 Allowing multiple choices

None of the tk, tix, and ttk widgets allow for multiple choices. You can implement a widget with multiple choices by providing a list of independent check boxes. A Labelframe would help organize that list of check boxes. Then your python program can collect all the choices into a list.

The Center for Disease Control provides tables of life expectancy based on both age and race. In our multi-cultural society many users prefer selecting more than one race, so the technique of this section could offer a solution in your attempt to expand upon the life expectancy program provided in this chapter.

Grid Geometry Manager

and More Widgets

4.1

This chapter introduces the grid geometry manager which is considered to be the most useful of the three available geometry managers for tkinter: pack, grid, and place. The first three chapters of this book used the pack geometry manager, so you already have considerable exposure to the pack geometry manager. Programs using the place geometry manager must specify the exact location for each widget. Not surprisingly most resources for geometry managers encourage readers to avoid using the place geometry manager. This book that advise. It leaves the place geometry for advanced study by the reader.

A special warning is included with every discussion of geometry managers. Never mix geometry managers within the same grouping of widgets. To be specific do not use both pack and grid when the two widgets have the same parent. You could use pack on one group of widgets and grid on another group of widgets all within the same window as long as those two groups of widgets do not share the same parent widget. Almost all example programs stick to using the grid. They never use two geometry managers. Obviously this is an advanced topic.

The grid geometry lays out the panel with a flexible grid of imaginary horizontal and vertical lines. Each location has a row and column number. All the python programs I've seen use positive integers beginning either with 0, 1, or 10. You can begin with any position integer. You can skip numbers. But, the grid geometry manager will organize your widgets left to right following the ascending order of your location numbers. The same rules hold true for the column numbers.

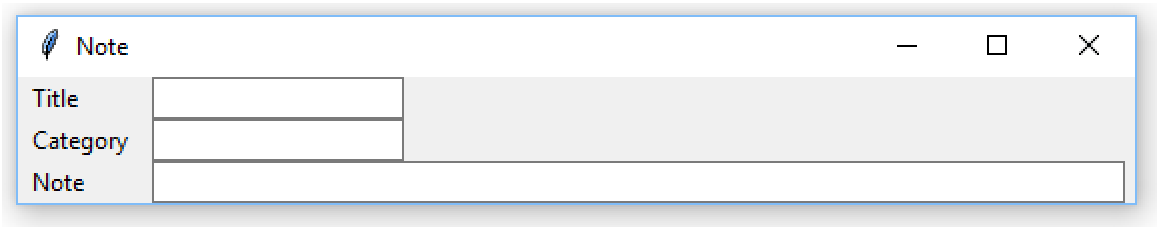
The widest widget in a column determines the width of the entire column. Likewise the tallest widget in a row determines the height of a row.

This example program demonstrates the placement of widgets using the grid geometry manager. This program generates the GUI for a program to maintain a list of brief one line notes.

```
# one line note pad

import tkinter as tk
```

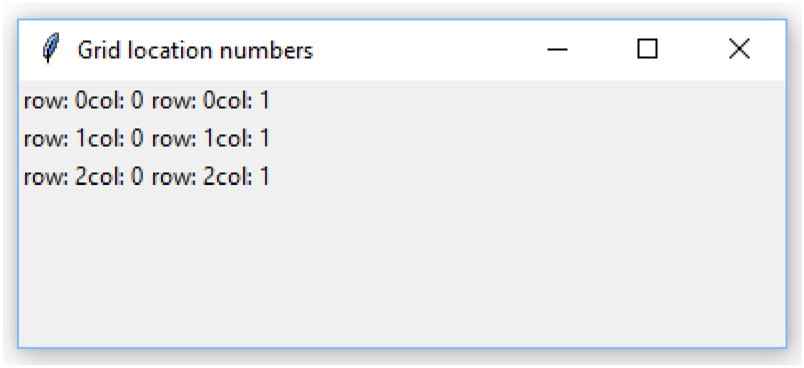
```
from tkinter import ttk

class App(ttk.Frame):

    """The adders gui and functions."""
    def __init__(self, parent, *args, **kwargs):
        ttk.Frame.__init__(self, parent, *args, **kwargs)
        self.root = parent
        self.init_gui(parent)

    def init_gui(self, master):
        """Builds GUI."""
        self.root.title('Note')
        self.titleLbl = ttk.Label(master, text="Title")
        self.titleLbl.grid(row=0, column=0, padx=5, sticky=tk.W)
        self.title = tk.StringVar()
        self.titleEntry = ttk.Entry(master, textvariable=self.title)
        self.titleEntry.grid(row=0,column=1, padx=5, sticky=tk.W)

        self.categoryLbl = ttk.Label(master, text="Category")
        self.categoryLbl.grid(row=1, column=0, padx=5, sticky=tk.W)
        self.categories = tk.StringVar()
        self.categoriesEntry = ttk.Entry(master, textvariable=self.categories)
        self.categoriesEntry.grid(row=1,column=1, padx=5, sticky=tk.W)

        self.noteLbl = ttk.Label(master, text="Note")
        self.noteLbl.grid(row=2, column=0, padx=5, sticky=tk.W)
        self.note = tk.StringVar()
        self.noteEntry = ttk.Entry(master, width=80, textvariable=self.note)
        self.noteEntry.grid(row=2,column=1, padx=5, sticky=tk.W)

if __name__ == '__main__':
    root = tk.Tk()
    App(root)
    root.mainloop()
```

Notice the grid calls use rows 0, 1, and 2 to identify the three rows. Of course you could use rows 1, 2, and 3, or rows 10, 20, and 30. The columns used are 0 and 1 because only two columns are used. The widest label is "Category," so the entire first column has that width. Likewise the entry field for the note is set to hold 80 characters while the default width for the title and category are both smaller. So the entire second column is much wider than the label column.



The next program displays a grid of labels each identified with its own row and column numbers for greater clarity in understanding the numbering system.

```
# Display a grid of labels with location numbers shown

import tkinter as tk
from tkinter import ttk

class App(ttk.Frame):

    """The adders gui and functions."""
    def __init__(self, parent, *args, **kwargs):
        ttk.Frame.__init__(self, parent, *args, **kwargs)
        self.root = parent
        self.init_gui(parent)

    def init_gui(self, master):
        """Builds GUI."""
        self.root.title('Grid location numbers')
        for rowN in range(3):
            for columnN in range(2):
                name = 'row: ' + str(rowN) + 'col: ' + str(columnN)
                self.lbl = ttk.Label(master, text=name)
                self.lbl.grid(row=rowN, column=columnN, sticky=tk.W)

if __name__ == '__main__':
    root = tk.Tk()
    App(root)
    root.mainloop()
```



```
Grid location numbers
row: 0col: 0 row: 0col: 1
row: 1col: 0 row: 1col: 1
row: 2col: 0 row: 2col: 1
```

The grid geometry manager allows the python programmer to quickly layout a GUI using the available tk and ttk widgets.

4.2 Widget collections

Another group of widgets is the tix collection. Other groups have developed numerous other collections of widgets.

This book focuses on the ttk widget collection plus the tk widgets which do not have displays opposite to the operating system they're used with.

You can create your own widgets and produce your own collection of widgets. Enjoy the power that TCL provides.

4.3 ttk.LabelFrame

The next step in the development of this python GUI program is to add control buttons such as SAVE, FIND, DELETE, and NEXT. So, our GUI will have two separate groups of widgets. For each group we will set up a LabelFrame which will add a label to the group and surround the group of widgets with a box. Thus users can easily grasp the organization of the GUI.

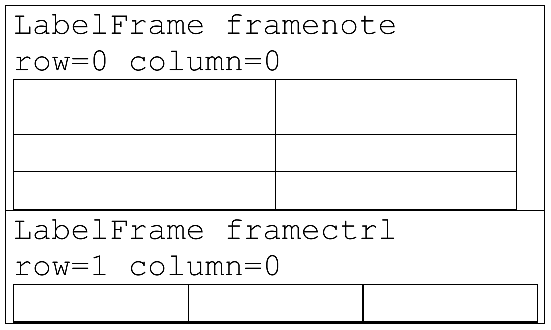
The command root = tk.Tk() calls the initialization method of the Tk class and returns a frame. That frame object is then passed to this class' initialization method when the command app = NoteApp(root) is executed. The first line of the initialization method, def __init__(self, master), receives that frame with the name of master.

Notice when we create the LabelFrame object by using the command, self.framenote = ttk.LabelFrame(master, width=180, height=40, text="Current Note"), that command has attached the new LabelFrame to the Frame object currently referenced as master. So, master is the parent of the LabelFrame referenced by framenote.
The labels and entry fields are attached to framenote, not master.

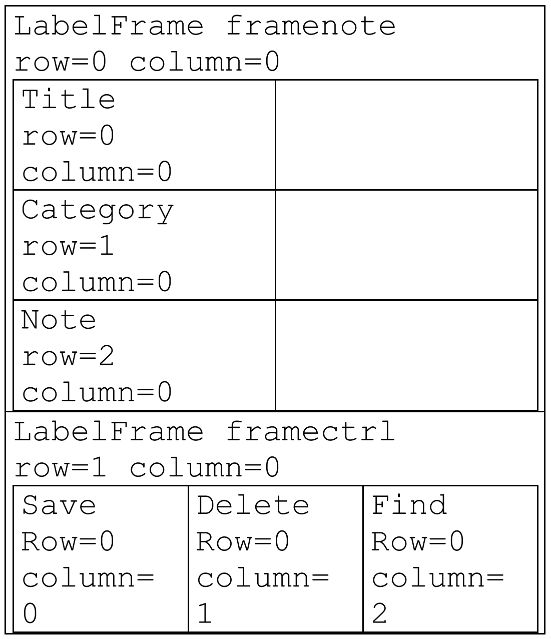
```
            self.titleLbl = ttk.Label(self.framenote, text="Title")
        self.titleLbl.grid(row=0, column=0, padx=5, sticky=tk.W)
```

Thus we have developed a family tree where master is the parent of framenote and framenote is the parent of the label referenced by the variable named titleLbl. As a result master has an instance of the grid geometry manager for placing the LabelFrame widgets.

The first LabelFrame widget named framenote is placed in location with row = 0 and column = 0. The second LabelFrame widget named framectrl is placed below the previous LabelFrame in the location with row = 1 and column = 0.



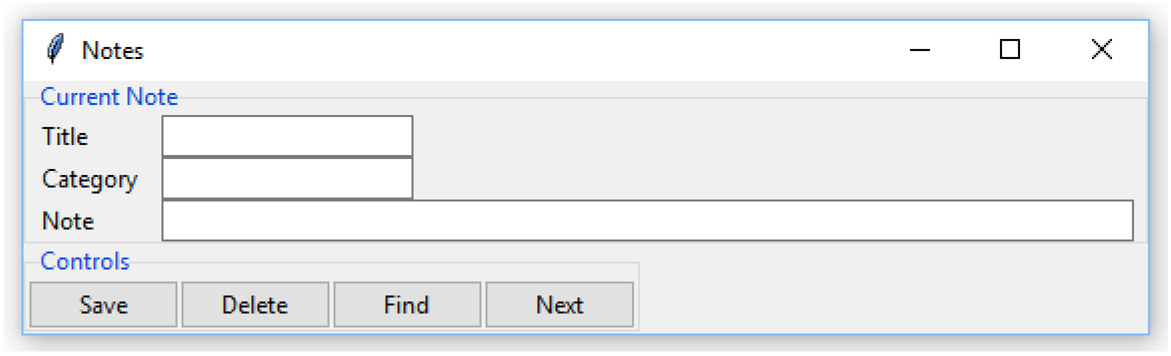
A separate instance of the grid geometry manager is set up for the LabelFrame widget, so when the label titleLbl is added to the LabelFrame widget we are using a different grid. Each frame will have its own instance of the grid geometry manager. Likewise the category label and entry fields will be set up in row 1 and the note label an entry fields will be set up in row 2.



Our GUI has three separate instances of the grid geometry manager. The first instance places widgets such as LabelFrames in the frame named manager. The second instance of the grid geometry manager places widgets such as the labels title, category, and note within the labelframe named framenote. The third instance of the grid geometry manager places widgets such as the buttons SAVE, DELETE, FIND, and Next within the labelframe named framectrl. Each instance of the grid geometry manager has its own grid with locations starting from row=0 and column=0. Thus on the chart shown above you will notice the location row=0 and column=0 appear three times, once for each instance of the grid geometry manager.

```
import tkinter as tk
from tkinter import ttk
from tkinter import *

class NoteApp(ttk.Frame):
    """This class creates packed frames for the GUI"""
    def __init__(self, master):
        ttk.Frame.__init__(self, master)

        master.title("Notes")
        # add label frame for note fields
        self.framenote = ttk.LabelFrame(master, width=180, height=40, text="Current Note")
        self.titleLbl = ttk.Label(self.framenote, text="Title")
        self.titleLbl.grid(row=0, column=0, padx=5, sticky=tk.W)
        self.title = tk.StringVar()
        self.titleEntry = ttk.Entry(self.framenote, textvariable=self.title)
        self.titleEntry.grid(row=0,column=1, padx=5, sticky=tk.W)

        self.categoryLbl = ttk.Label(self.framenote, text="Category")
        self.categoryLbl.grid(row=1, column=0, padx=5, sticky=tk.W)
        self.categories = tk.StringVar()
        self.categoriesEntry = ttk.Entry(self.framenote, textvariable=self.categories)
        self.categoriesEntry.grid(row=1,column=1, padx=5, sticky=tk.W)

        self.noteLbl = ttk.Label(self.framenote, text="Note")
        self.noteLbl.grid(row=2, column=0, padx=5, sticky=tk.W)
        self.note = tk.StringVar()
        self.noteEntry = ttk.Entry(self.framenote, width=80, textvariable=self.note)
        self.noteEntry.grid(row=2,column=1, padx=5, sticky=tk.W)

        self.framenote.grid(row=0, column=0)

        # add label frame for control buttons
        self.framectrl = ttk.LabelFrame(master, width=180, height=40, text="Controls")
        self.ctrlBtnSave = ttk.Button(self.framectrl, text="Save")
        self.ctrlBtnSave.grid(row=0, column=1)
        self.ctrlBtnDlt = ttk.Button(self.framectrl, text="Delete")
        self.ctrlBtnDlt.grid(row=0, column=2)
        self.ctrlBtnFind = ttk.Button(self.framectrl, text="Find")
        self.ctrlBtnFind.grid(row=0, column=3)
        self.ctrlBtnNext = ttk.Button(self.framectrl, text="Next")
        self.ctrlBtnNext.grid(row=0, column=4)
        self.framectrl.grid(row=1, column=0, sticky=tk.W)

root = tk.Tk()
app = NoteApp(root)
root.mainloop()
```

This program produces this GUI.

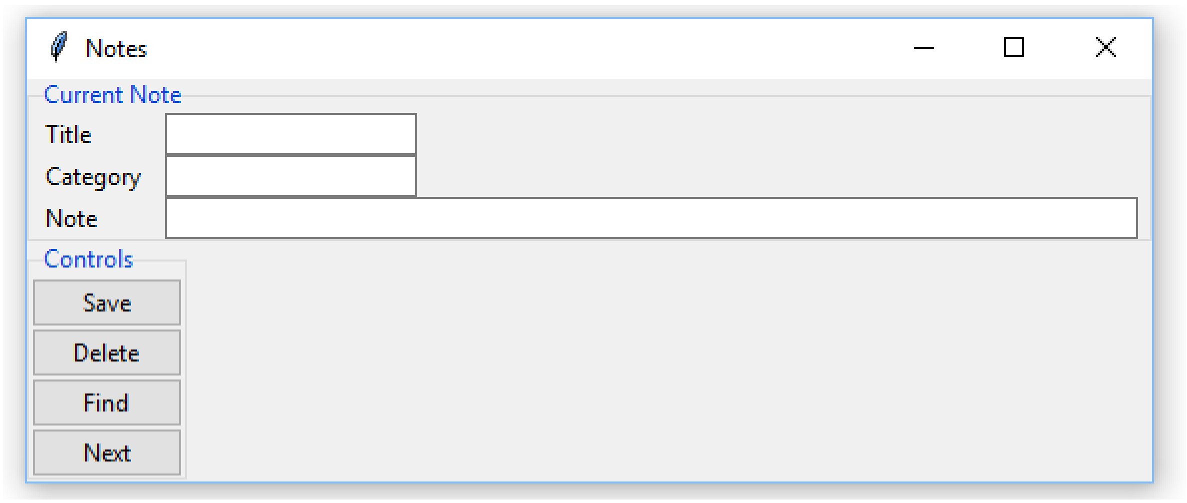


Aside:

The buttons attached to framectrl could be attached using the pack geometry manager even though the grid geometry manager is used for the other frames.

```
import tkinter as tk
from tkinter import ttk
from tkinter import *

class NoteApp(ttk.Frame):
    """This class creates packed frames for the GUI"""

    def __init__(self, master):
        ttk.Frame.__init__(self, master)

        master.title("Notes")
        # add label frame for note fields
        self.framenote = ttk.LabelFrame(master, width=180, height=40, text="Current Note")
        self.titleLbl = ttk.Label(self.framenote, text="Title")
        self.titleLbl.grid(row=0, column=0, padx=5, sticky=tk.W)
        self.title = tk.StringVar()
        self.titleEntry = ttk.Entry(self.framenote, textvariable=self.title)
        self.titleEntry.grid(row=0,column=1, padx=5, sticky=tk.W)

        self.categoryLbl = ttk.Label(self.framenote, text="Category")
        self.categoryLbl.grid(row=1, column=0, padx=5, sticky=tk.W)
        self.category = tk.StringVar()
        self.categoryEntry = ttk.Entry(self.framenote, textvariable=self.category)
        self.categoryEntry.grid(row=1,column=1, padx=5, sticky=tk.W)

        self.noteLbl = ttk.Label(self.framenote, text="Note")
        self.noteLbl.grid(row=2, column=0, padx=5, sticky=tk.W)
        self.note = tk.StringVar()
        self.noteEntry = ttk.Entry(self.framenote, width=80, textvariable=self.note)
        self.noteEntry.grid(row=2,column=1, padx=5, sticky=tk.W)
        self.framenote.grid(row=0, column=0)

        # add label frame for control buttons
        self.framectrl = ttk.LabelFrame(master, width=180, height=40, text="Controls")
        self.ctrlBtnSave = ttk.Button(self.framectrl, text="Save", command=self.save_note)
        self.ctrlBtnSave.pack()
        self.ctrlBtnDlt = ttk.Button(self.framectrl, text="Delete", command=self.delete_note)
        self.ctrlBtnDlt.pack()
        self.ctrlBtnFind = ttk.Button(self.framectrl, text="Find", command=self.find_note)
        self.ctrlBtnFind.pack()
        self.ctrlBtnNext = ttk.Button(self.framectrl, text="Next", command=self.next_note)
        self.ctrlBtnNext.pack()
        self.framectrl.grid(row=1, column=0, sticky=tk.W)

    def save_note(self):
        print("pressed save")
        self.notes[(self.title.get(),self.category.get())] = self.note.get()
        print("all notes")
        for (t,c) in self.notes:
            print("t: " + t + " c: " + c)

    def delete_note(self):
        print("pressed delete")
        del self.notes[(self.title.get(),self.category.get())]

    def find_note(self):
        print("pressed find")
        self.targetcategory = self.category.get()
        self.notelist = [ (t,c) for (t,c) in self.notes if c == self.targetcategory ]
        if len(self.notelist) > 0:
            (t0, c0) = self.notelist[0]
            self.title.set(t0)
            self.note.set(self.notes[(t0,c0)])

    def next_note(self):
        print("pressed next")



root = tk.Tk()
app = NoteApp(root)
app.notes = dict()
app.notelist = list()
root.mainloop()
```

This program produces this GUI.



I do not encourage doing that. You could easily make a mistake and face an error message. Suppose I changed this command,

```
self.framectrl.grid(row=1, column=0, sticky=tk.W)
```

to also use the pack geometry manager. So the new command would be

```
self.framectrl.pack()
```

That is an easy mistake to make. You would face the following error message on the console.

```
 RESTART: C:/Users/John/Documents/Python/Tkinter Book/C4 Grid/notes
program/multiframeErrorMixGridAndPackTtkV5.py
Traceback (most recent call last):
  File "C:/Users/John/Documents/Python/Tkinter Book/C4 Grid/notes
program/multiframeErrorMixGridAndPackTtkV5.py", line 77, in <module>
    app = NoteApp(root)
  File "C:/Users/John/Documents/Python/Tkinter Book/C4 Grid/notes
program/multiframeErrorMixGridAndPackTtkV5.py", line 49, in __init__
    self.framectrl.pack()
  File "C:\Users\John\AppData\Local\Programs\Python\Python35-32\lib\tkinter\__init__.py",
line 1990, in pack_configure
    + self._options(cnf, kw))
_tkinter.TclError: cannot use geometry manager pack inside . which already has slaves
managed by grid
>>>
```

This error messages traces the important commands leading to the error. On line 77 of the program the command app=NoteApp(root) called the initialization or init function for the NoteApp class. While processing those commands an error occurred during the execution of the command self.framectrl.pack(). We made that pack mistake, so we expected that command to be part of the error. That pack command was sent to the Tcl interpreter which produced an error message. The key part of the message is "TclError: cannot use geometry manager pack inside . Which already has slaves managed by grid." Apparently this erroneous usage of the pack command went to the Tcl interpreter, which enforced the Tcl rules. The dot (.) means the parent widget of the current widget, which is named framectrl. That widget's parent is called self because it is the object produced by execution of the init function. Also notice the program has this command self.framectrl.pack() and previously the command self.framenote.grid(row=0, column=0). Thus both grid and pack have been used to connect widgets with self. That's an error. The Tcl interpreter noticed that error and correctly produced an error message. Everyone wishes the error messages were easier to read and understand, but the Tcl interpreter does not understand the python program, so the Tcl interpreter can only display standard error messages for our breaking Tcl rules.

Fortunately most users of Tcl quickly learn to identify the errors. With practice your understanding of these arcane error messages will improve. You will never learn to love Tcl error messages.

The final decision is simple. You should use the grid geometry manager for all your work.

4.4 tk.MessageBox

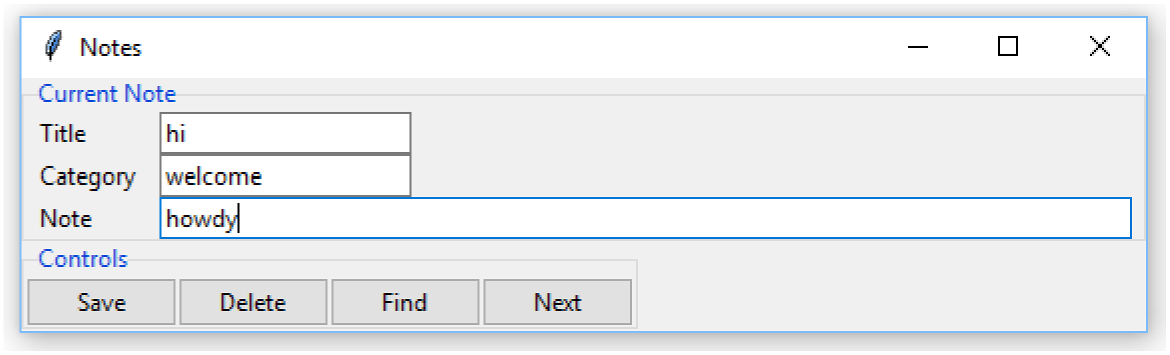
Suppose the program is unable to find any notes when searching by the requested category. Then the program should display a message warning the user and explaining the situation. You can use tk's messagebox to alert the user. The code is shown below.

```
def find_note(self):
    print("pressed find")
    self.targetcategory = self.category.get()
    self.notelist = [ (t,c) for (t,c) in self.notes if c == self.targetcategory ]
    if len(self.notelist) > 0:
        (t0, c0) = self.notelist[0]
        self.title.set(t0)
        self.note.set(self.notes[(t0,c0)])
    else:
        tk.messagebox.showinfo("Error", "No matching notes found for that category")
```

Thus when there are no notes in the list the else condition is executed. Thus this command is run.

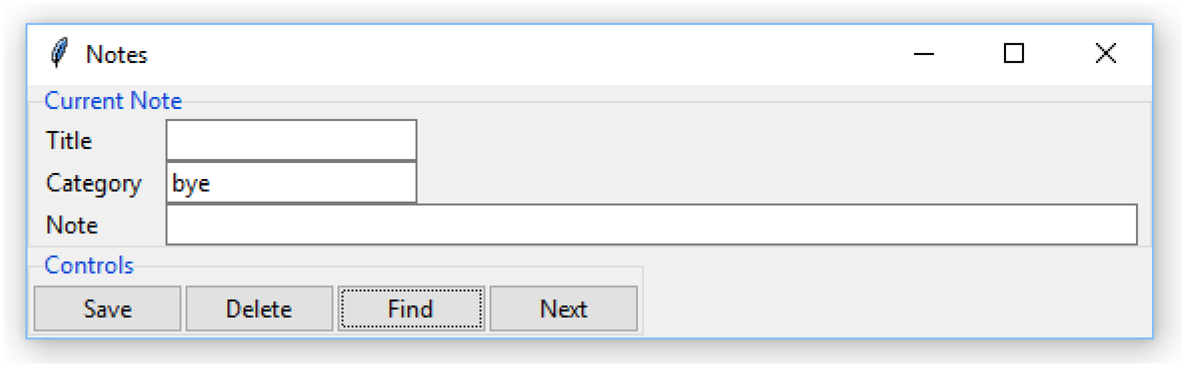
```
tk.messagebox.showinfo("Error", "No matching notes found for that category")
```

Before testing that command I should set up a note in the dictionary named notes.



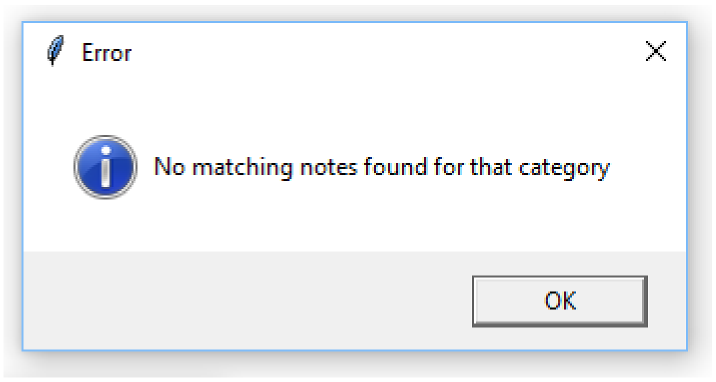
Title: hi
Category: welcome
Note: howdy

So the dictionary has only one note. That note is in the category named "welcome." I press the SAVE button. Next I delete both the title and the note from the display. I enter the category of "Bye," so I expect no matches.



Title:
Category: bye
Note:

Then I press the FIND button. That command will produce an alert as shown below.


Error

No matching notes found for that category

That message does not follow the look of a Windows message. Another possibility is to use ttk.ttkSimpleDialog widget. Being a ttk widget it will follow the look of the supporting operating system. That widget is not currently included in the ttk package, so it must be loaded separately.

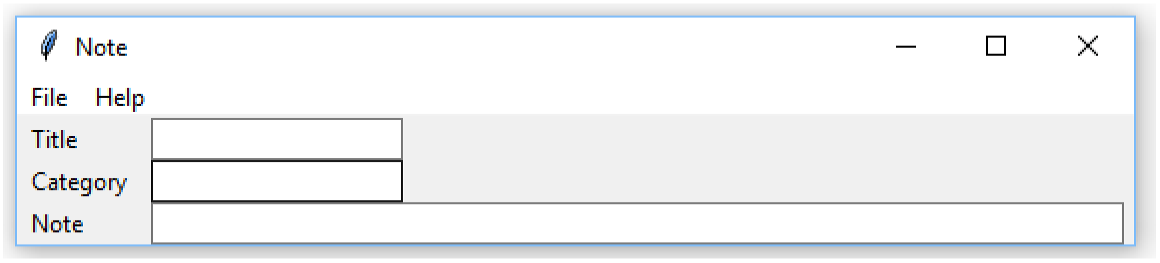
5.1 Using a menu

Instead of cluttering your GUI window with control buttons you can collect and move all those control buttons to a menu by using the menu widget. For the example of a python program to keep notes your python program might have a File menu with an OPEN, a SAVE, and a NEW option. Also the END or EXIT PROGRAM option is usually under the FILE menu. Thus easily four buttons are removed from the screen and hidden in the FILE menu until needed. Likewise most applications have a HELP menu option. Wikipedia has an entry explaining menubars at https://en.wikipedia.org/wiki/Menu_bar.

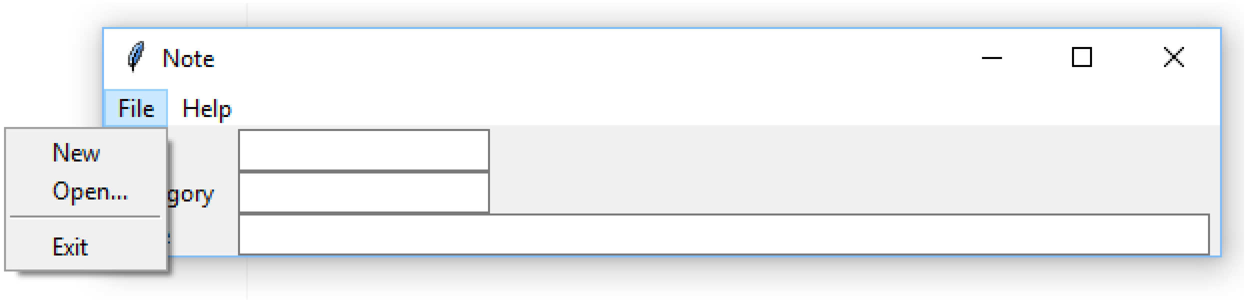
Next we will add a menubar to our python program to keep notes. The source code for the GUI portion is shown below.

```
# one line note pad

import tkinter as tk
from tkinter import ttk
from tkinter import *

class NoteApp(ttk.Frame):

    """The adders gui and functions."""
    def __init__(self, parent, *args, **kwargs):
        ttk.Frame.__init__(self, parent, *args, **kwargs)
        self.root = parent
        self.init_gui(parent)

    def new_file(self):
        print ("Create a new file.")

    def open_file(self):
        print("Open file.")

    def about(self):
        print("There is no help.")

    def init_gui(self, master):
        """Builds GUI."""
        self.root.title('Note')

        self.menubar = Menu(master)
        master.config(menu=self.menubar)
        self.file_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="File", menu=self.file_menu)
        self.file_menu.add_command(label="New", command=self.new_file)
        self.file_menu.add_command(label="Open...", command=self.open_file)
        self.file_menu.add_separator()
        self.file_menu.add_command(label="Exit", command=self.master.destroy)

        self.help_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Help", menu=self.help_menu)
        self.help_menu.add_command(label="About...", command=self.about)

        self.titleLbl = ttk.Label(master, text="Title")
        self.titleLbl.grid(row=0, column=0, padx=5, sticky=tk.W)
        self.title = tk.StringVar()
        self.titleEntry = ttk.Entry(master, textvariable=self.title)
        self.titleEntry.grid(row=0,column=1, padx=5, sticky=tk.W)

        self.categoryLbl = ttk.Label(master, text="Category")
        self.categoryLbl.grid(row=1, column=0, padx=5, sticky=tk.W)
        self.categories = tk.StringVar()
        self.categoriesEntry = ttk.Entry(master, textvariable=self.categories)
        self.categoriesEntry.grid(row=1,column=1, padx=5, sticky=tk.W)

        self.noteLbl = ttk.Label(master, text="Note")
        self.noteLbl.grid(row=2, column=0, padx=5, sticky=tk.W)
        self.note = tk.StringVar()
        self.noteEntry = ttk.Entry(master, width=80, textvariable=self.note)
        self.noteEntry.grid(row=2,column=1, padx=5, sticky=tk.W)

if __name__ == '__main__':
    root = tk.Tk()
    NoteApp(root)
    root.mainloop
```

Notice the functions to create a new file, open a new file, and exit the program are simply stubs. We will expand those functions later. For now we can run this python program using IDLE and create the window shown below.



The new menubar appears at the top of the window under the window's title. This menubar has two drop down menus named File and Help. Pressing the File option produces this drop down menu.



When you move the mouse over the File option the drop down menu appears. It shows the options of New, Open, and Exit. They function like buttons. When you add the new option you can also add a command to call a python function like you did for a button.

```
self.file_menu.add_command(label="New", command=self.new_file)
self.file_menu.add_command(label="Open...", command=self.open_file)
self.file_menu.add_separator()
self.file_menu.add_command(label="Exit", command=self.master.destroy)
```

The add separator function simply display a line separating the FILE and OPEN options from the EXIT option. That's simply for decoration since some operating systems customarily include those separators. It is not required.

Before executing those commands the variable named file_menu must be created.

```
self.file_menu = Menu(self.menubar, tearoff=0)
```

Tear off menus which can float free from the drop down menu are supported on Unix systems, but not Windows systems. Since this book was written on a Windows machines the option tearoff is set to zero. Again before using that command we must create another variable named menubar.

```
self.menubar = Menu(master)
master.config(menu=self.menubar)
```

This first command creates the variable named menubar. It points to an instance of a menu widget. That menu widget is a slave (subordinate) to the widget named master. Of course master is the root passed to the init function. So master is a Tk.Frame widget which had been created by the following command.

```
root = tk.Tk()
```

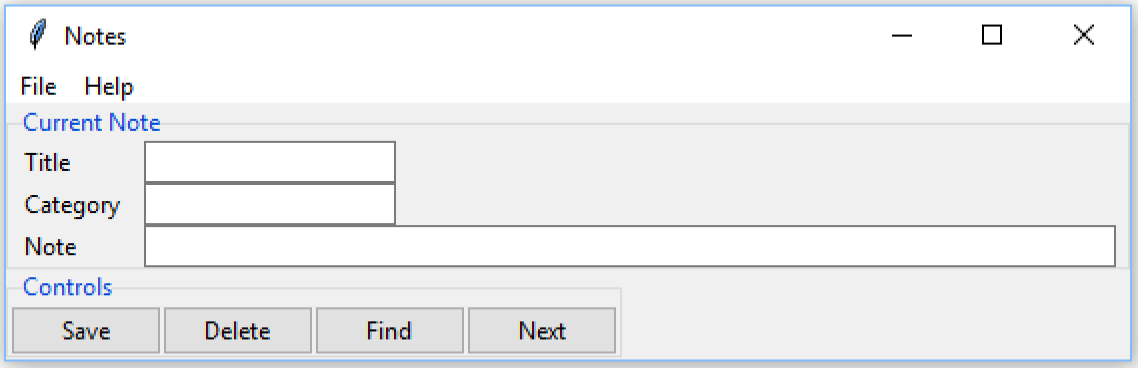
In chapter one you were introduced to the config command. This command sets the menu for the frame to be menubar.

Of course that should have been a ttk frame, not a tk frame, so the window would more properly conform to the supporting operating systems window theme

Now we are challenged with combining this new GUI with our python program. This python program shown below is a combination of the simple menu program together with the one line note program from chapter four.

```
import tkinter as tk
from tkinter import ttk
from tkinter import *
from tkinter.filedialog import askopenfilename

class NoteApp(ttk.Frame):
    """This class creates packed frames for the GUI"""

    def __init__(self, master):
        ttk.Frame.__init__(self, master)
        self.root = master
        self.init_gui(master)

    def init_gui(self, master):
        master.title("Notes")
        # Create menubar
        self.menubar = Menu(master)
        master.config(menu=self.menubar)
        self.file_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="File", menu=self.file_menu)
        self.file_menu.add_command(label="New", command=self.new_file)
        self.file_menu.add_command(label="Open...", command=self.open_file)
        self.file_menu.add_separator()
        self.file_menu.add_command(label="Exit", command=self.master.destroy)

        self.help_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Help", menu=self.help_menu)
        self.help_menu.add_command(label="About...", command=self.about)
        # add label frame for note fields
        self.framenote = ttk.LabelFrame(master, width=180, height=40, text="Current Note")
        self.titleLbl = ttk.Label(self.framenote, text="Title")
        self.titleLbl.grid(row=0, column=0, padx=5, sticky=tk.W)
        self.title = tk.StringVar()
        self.titleEntry = ttk.Entry(self.framenote, textvariable=self.title)
        self.titleEntry.grid(row=0,column=1, padx=5, sticky=tk.W)

        self.categoryLbl = ttk.Label(self.framenote, text="Category")
        self.categoryLbl.grid(row=1, column=0, padx=5, sticky=tk.W)
        self.category = tk.StringVar()
        self.categoryEntry = ttk.Entry(self.framenote, textvariable=self.category)
        self.categoryEntry.grid(row=1,column=1, padx=5, sticky=tk.W)

        self.noteLbl = ttk.Label(self.framenote, text="Note")
        self.noteLbl.grid(row=2, column=0, padx=5, sticky=tk.W)
        self.note = tk.StringVar()
        self.noteEntry = ttk.Entry(self.framenote, width=80, textvariable=self.note)
        self.noteEntry.grid(row=2,column=1, padx=5, sticky=tk.W)

        self.framenote.grid(row=0, column=0)

        # add label frame for control buttons
        self.framectrl = ttk.LabelFrame(master, width=180, height=40, text="Controls")
        self.ctrlBtnSave = ttk.Button(self.framectrl, text="Save", command=self.save_note)
        self.ctrlBtnSave.grid(row=0, column=1)
        self.ctrlBtnDlt = ttk.Button(self.framectrl, text="Delete", command=self.delete_note)
        self.ctrlBtnDlt.grid(row=0, column=2)
        self.ctrlBtnFind = ttk.Button(self.framectrl, text="Find", command=self.find_note)
        self.ctrlBtnFind.grid(row=0, column=3)
        self.ctrlBtnNext = ttk.Button(self.framectrl, text="Next", command=self.next_note)
        self.ctrlBtnNext.grid(row=0, column=4)
        self.framectrl.grid(row=1, column=0, sticky=tk.W)

        self.menubar = Menu(master)
        master.config(menu=self.menubar)
        self.file_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="File", menu=self.file_menu)
        self.file_menu.add_command(label="New", command=self.new_file)
        self.file_menu.add_command(label="Open...", command=self.open_file)
        self.file_menu.add_separator()
        self.file_menu.add_command(label="Exit", command=self.master.destroy)

        self.help_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Help", menu=self.help_menu)
        self.help_menu.add_command(label="About...", command=self.about)

    def new_file(self):
        print ("Create a new file.")

    def open_file(self):
        print("Open file.")

    def about(self):
        print("There is no help.")
    def save_note(self):
        print("pressed save")
        self.notes[(self.title.get(),self.category.get())] = self.note.get()
        print("all notes")
        for (t,c) in self.notes:
            print("t: " + t + " c: " + c)

    def delete_note(self):
        print("pressed delete")
        del self.notes[(self.title.get(),self.category.get())]

    def find_note(self):
        print("pressed find")
        self.targetcategory = self.category.get()
        self.notelist = [ (t,c) for (t,c) in self.notes if c == self.targetcategory ]
        if len(self.notelist) > 0:
            (t0, c0) = self.notelist[0]
            self.title.set(t0)
            self.note.set(self.notes[(t0,c0)])
        else:
            tk.messagebox.showinfo("Error", "No matching notes found for that category")
    def next_note(self):
        print("pressed next")



root = tk.Tk()
app = NoteApp(root)
app.notes = dict()
app.notelist = list()
root.mainloop()
```

The GUI for this program is shown below. Almost none of the supporting python functions work yet.



The control buttons for Save, Delete, Find, and Next shown the current note have terrible names. The Save button kept the current note in the dictionary, so Keep would be a better name. Save is too often used with saving files, thus confusing the user. That button was set by this command.

```
self.ctrlBtnSave = ttk.Button(self.framectrl, text="Save", command=self.save_note)
```

That command becomes as shown here.

```
self.ctrlBtnSave = ttk.Button(self.framectrl, text="Keep", command=self.keep_note)
```

Notice the function name was also changed from save_note to keep_note. That function is now defined by this command.

```
def keep_note(self):
    print("pressed keep note")
    self.notes[(self.title.get(),self.category.get())] = self.note.get()
    print("all notes")
    for (t,c) in self.notes:
        print("t: " + t + " c: " + c)
```

When the program is thoroughly checked this function will be slimmed down to keep the note in the dictionary and do nothing more. Then the function will be as shown here.

```
def keep_note(self):
            self.notes[(self.title.get(),self.category.get())] = self.note.get()
```

The keep_note function should also set a dirty bit. Wikipedia has a brief description of the purpose of a dirty bit at https://en.wikipedia.org/wiki/Dirty_bit. The dirty bit is set when the computer memory is changed. When that memory is saved in a file the dirty bit is reset to zero. For our purposes the dirty bit is set when a note is kept in the diction of notes. When the user presses the Save option and saves all the notes to a file that dirty bit is reset to 0. Thus two functions are impacted. The keep notes function is expanded to set the dirty bit as shown below.

```
def keep_note(self):
    print("pressed keep note")
    self.notes[(self.title.get(),self.category.get())] = self.note.get()
    self.dirty_bit = 1
    print("all notes")
    for (t,c) in self.notes:
        print("t: " + t + " c: " + c)
```

The save function will reset that dirty bit as shown below.

Surprisingly that save function is not even defined yet. Here it that new save file function.

```
    def save_file(self):
        print("Save file was pressed.")
        self.dirty_bit = 0
```

Obviously a lot more work needs to be completed on that save file function. Next the save file option must be added to File's cascade of options.

```
        self.file_menu.add_command(label="Save", command=self.save_file)
```

Also the label for the controls section should be Note Controls, not briefly Controls so users would not be confused. That label was set by this command.

```
                self.framectrl = ttk.LabelFrame(master, width=180, height=40, text="Controls")
```

So, that command becomes.

```
                self.framectrl = ttk.LabelFrame(master, width=180, height=40, text="Note Controls")
```

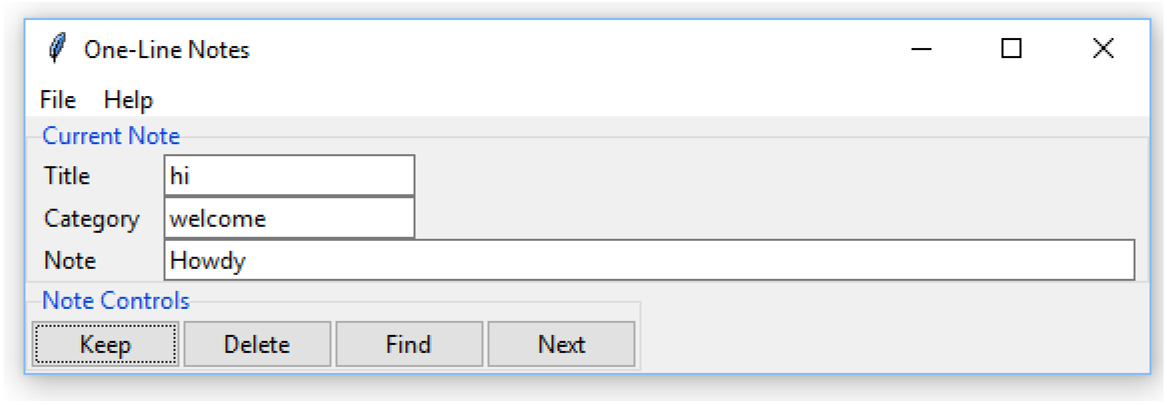
Finally the title for the window should be "One Line Notes," not "Notes." That was set by this command.

```
                master.title("Notes")
```

So, the following command to set the title of the frame should be

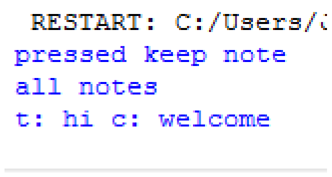
```
        master.title("One-Line Notes")
```

Next we run the program again. The GUI contains most of our changes. The window's title is no "One-Line Notes." The label for the label frame for the note controls is now "Note Controls" and the first control button is labelled "Keep," not "Save."



hi
welcome
Howdy

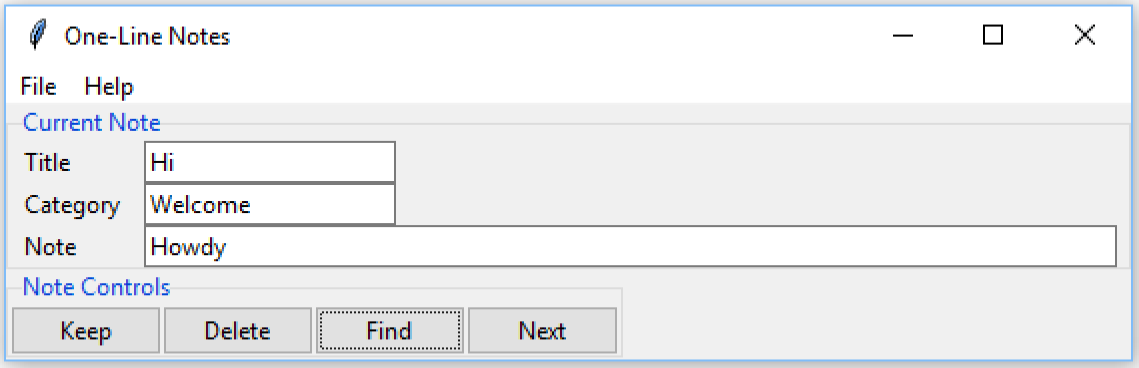
I typed values for each of the three input fields and pressed the Keep button, so these messages appeared on the console.



```
 RESTART: C:/Users/
pressed keep note
all notes
t: hi c: welcome
```

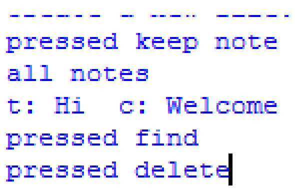
Those are the messages we expected to have displayed, so the program appears to be running properly.

Next I must check the Keep and Find control buttons. I cleared the title and note entries. Pressing the Find control button the GUI becomes as shown here.



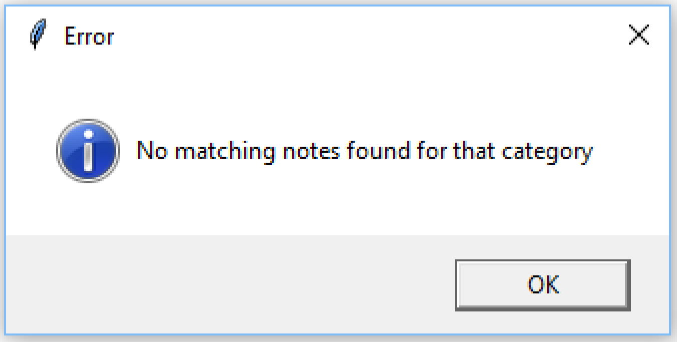
Title: Hi
Category: Welcome
Note: Howdy

So the find function worked properly. However the find function only matches the category value. Next I test the delete control button. This message appears on console “pressed delete.” The dictionary should now be empty.



```
pressed keep note
all notes
t: Hi  c: Welcome
pressed find
pressed delete
```

Let's try to find a note in that empty dictionary. Again I cleared title and note fields. Then I pressed the find control button again.


Error

No matching notes found for that category

An error message appears in a message box above my window. There were no matching notes, so the delete function worked successfully. Another concern is that the dirty bit for the dictionary is still set to 1 even though that invocation of the delete function emptied that dictionary. A quick solution is to check the length of that dictionary.

```
 def delete_note(self):
   print("pressed delete")
   del self.notes[(self.title.get(),self.category.get())]
   if len(self.notes) == 0:
      self.dirty_bit = 0
```

Now we must write the missing next_note function.

```
def find_note(self):
   print("pressed find")
   self.targetcategory = self.category.get()
   self.notelist = [ (t,c) for (t,c) in self.notes if c == self.targetcategory ]
   if len(self.notelist) > 0:
      (t0, c0) = self.notelist[0]
      self.title.set(t0)
      self.note.set(self.notes[(t0,c0)])
      self.notelist.remove((t0, c0))
   else:
      tk.messagebox.showinfo("Error", "No matching notes found for that category")

def next_note(self):
   print("pressed next")
   if len(self.notelist) > 0:
      (t0, c0) = self.notelist[0]
      self.title.set(t0)
      self.note.set(self.notes[(t0,c0)])
      self.notelist.remove((t0, c0))
   else:
      tk.messagebox.showinfo("Error", "No more matching notes found for that category")
```

The simplest method is to remove the note from the list of matching notes as the note is displayed. That modification goes to the find_note function. The next_note function is then simply the last steps of the find_note function. It checks for an empty list and displays the first element of that list. Setting up those steps in a separate function would demonstrate better programming skill.

Now we face the process of opening and saving files. Here is an attempt at the code. Obviously this approach is not the best. Check the internet for a safer approach.

```
    def open_file(self):
##        print("Open file.")
        self.myfile = 
askopenfilename(initialdir="C:/Users/Batman/Documents/Programming/tkinter/",
                           filetypes =(("Text File", "*.txt"),("All Files","*.*")),
                           title = "Choose a file."
                           )
##        print("file: " + self.myfile)
        #Using try in case user types in unknown file or closes without choosing a file.
        try:
            with open(self.myfile,'r') as f:
##                print("opened file")
                self.notes = dict()
                self.dirty_bit = 0
                print("reset notes dictionary")
                for item in f:
                    if ':' in item:
                        key1, key2, value = item.split(':')
                        self.notes[(key1, key2)] = value
                    else:
                        print("Error reading item: " + item)
        except:
            print("No file exists")

    def save_file(self):
        print("Save file: " + self.myfile)
        self.dirty_bit = 0
        with open(self.myfile, 'w') as f:
            for key, value in self.notes.items():
                (key1, key2) = key
                f.write('%s:%s:%s\n' % (key1, key2, value))
```

Notice the dirty bit. When a file is read the notes dictionary is reset to empty, so the dirty bit is reset to zero. When the dictionary of notes is saved to a file then the dirty bit is also set to zero. Next we must rewrite the python function for the Exit button. Currently pressing the exit button destroys the windows.

```
    self.file_menu.add_command(label="Exit", command=self.master.destroy)
```

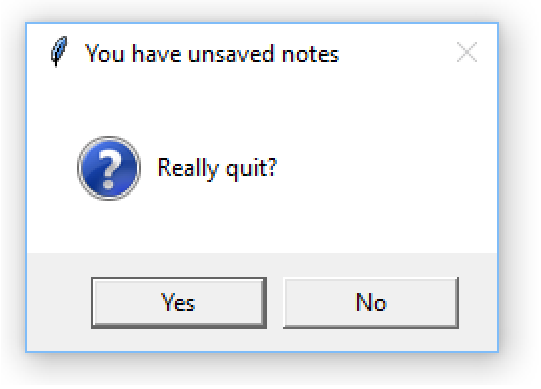
Let's check the dirty bit before eliminating the window.

```
        self.file_menu.add_command(label="Exit", command=self.exit_run)
```

Now to write the code for the exit function.

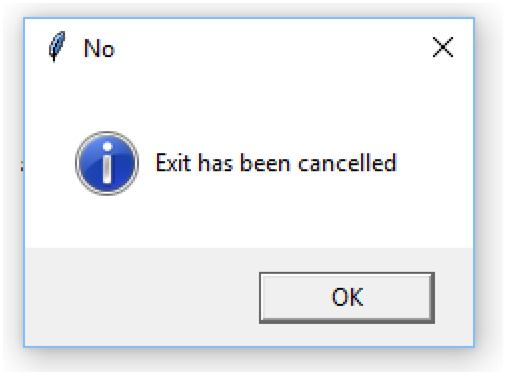
```
def exit_run(self):
  print("Exit_run dirty bit: " + str(self.dirty_bit))
  if self.dirty_bit == 1:
     if messagebox.askyesno('You have unsaved notes', 'Really quit?'):
        self.root.destroy()
     else:
        messagebox.showinfo('No', 'Exit has been cancelled')
  else:
     self.root.destroy()
```

Next we test that condition which produces this alert.


You have unsaved notes

Really quit?

If you choose the no option then you receive this informational alert.



Exit has been cancelled

```
Let's review the entire python program.
```

```
import tkinter as tk
from tkinter import ttk
from tkinter import *
from tkinter.filedialog import askopenfilename

class NoteApp(ttk.Frame):
    """This class creates packed frames for the GUI"""

    def __init__(self, master):
        ttk.Frame.__init__(self, master)
        self.root = master
        self.init_gui(master)

    def init_gui(self, master):
        master.title("One-Line Notes")
        # Create menubar
        self.menubar = Menu(master)
        master.config(menu=self.menubar)
        self.file_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="File", menu=self.file_menu)
        self.file_menu.add_command(label="Open...", command=self.open_file)
        self.file_menu.add_command(label="Save", command=self.save_file)
        self.file_menu.add_separator()
        self.file_menu.add_command(label="Exit", command=self.exit_run)

        self.help_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Help", menu=self.help_menu)
        self.help_menu.add_command(label="About...", command=self.about)
        # add label frame for note fields
        self.framenote = ttk.LabelFrame(master, width=180, height=40, text="Current Note")
        self.titleLbl = ttk.Label(self.framenote, text="Title")
        self.titleLbl.grid(row=0, column=0, padx=5, sticky=tk.W)
        self.title = tk.StringVar()
        self.titleEntry = ttk.Entry(self.framenote, textvariable=self.title)
        self.titleEntry.grid(row=0,column=1, padx=5, sticky=tk.W)
        self.categoryLbl = ttk.Label(self.framenote, text="Category")
        self.categoryLbl.grid(row=1, column=0, padx=5, sticky=tk.W)
        self.category = tk.StringVar()
        self.categoryEntry = ttk.Entry(self.framenote, textvariable=self.category)
        self.categoryEntry.grid(row=1,column=1, padx=5, sticky=tk.W)

        self.noteLbl = ttk.Label(self.framenote, text="Note")
        self.noteLbl.grid(row=2, column=0, padx=5, sticky=tk.W)
        self.note = tk.StringVar()
        self.noteEntry = ttk.Entry(self.framenote, width=80, textvariable=self.note)
        self.noteEntry.grid(row=2,column=1, padx=5, sticky=tk.W)

        self.framenote.grid(row=0, column=0)

        # add label frame for control buttons
        self.framectrl = ttk.LabelFrame(master, width=180, height=40, text="Note Controls")
        self.ctrlBtnSave = ttk.Button(self.framectrl, text="Keep", command=self.keep_note)
        self.ctrlBtnSave.grid(row=0, column=1)
        self.ctrlBtnDlt = ttk.Button(self.framectrl, text="Delete", command=self.delete_note)
        self.ctrlBtnDlt.grid(row=0, column=2)
        self.ctrlBtnFind = ttk.Button(self.framectrl, text="Find", command=self.find_note)
        self.ctrlBtnFind.grid(row=0, column=3)
        self.ctrlBtnNext = ttk.Button(self.framectrl, text="Next", command=self.next_note)
        self.ctrlBtnNext.grid(row=0, column=4)
        self.framectrl.grid(row=1, column=0, sticky=tk.W)

        self.menubar = Menu(master)
        master.config(menu=self.menubar)
        self.file_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="File", menu=self.file_menu)
        self.file_menu.add_command(label="Open", command=self.open_file)
        self.file_menu.add_command(label="Save", command=self.save_file)
        self.file_menu.add_separator()
        self.file_menu.add_command(label="Exit", command=self.exit_run)

        self.help_menu = Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="Help", menu=self.help_menu)
        self.help_menu.add_command(label="About...", command=self.about)


    def open_file(self):
##        print("Open file.")
        self.myfile =
askopenfilename(initialdir="C:/Users/Batman/Documents/Programming/tkinter/",
                        filetypes =(("Text File", "*.txt"),("All Files","*.*")),
                        title = "Choose a file."
                        )
##        print("file: " + self.myfile)
        #Using try in case user types in unknown file or closes without choosing a file.
        try:
            with open(self.myfile,'r') as f:
##                  print("opened file")
                self.notes = dict()
                self.dirty_bit = 0
                print("reset notes dictionary")
                for item in f:
                    if ':' in item:
                        key1, key2, value = item.split(':')
                        self.notes[(key1, key2)] = value
                    else:
                        print("Error reading item: " + item)
        except:
            print("No file exists")

    def save_file(self):
        print("Save file: " + self.myfile)
        self.dirty_bit = 0
        with open(self.myfile, 'w') as f:
            for key, value in self.notes.items():
                (key1, key2) = key
                f.write('%s:%s:%s\n' % (key1, key2, value))

    def exit_run(self):
        print("Exit_run dirty bit: " + str(self.dirty_bit))
        if self.dirty_bit == 1:
            if messagebox.askyesno('You have unsaved notes', 'Really quit?'):
                self.root.destroy()
            else:
                messagebox.showinfo('No', 'Exit has been cancelled')
        else:
            self.root.destroy()


    def about(self):
        print("There is no help.")

    def keep_note(self):
        print("pressed keep note")
        self.notes[(self.title.get(),self.category.get())] = self.note.get()
        self.dirty_bit = 1
        print("all notes")
        for (t,c) in self.notes:
            print("t: " + t + " c: " + c)

    def delete_note(self):
        print("pressed delete")
        del self.notes[(self.title.get(),self.category.get())]
        if len(self.notes) == 0:
            self.dirty_bit = 0

    def find_note(self):
        print("pressed find")
        self.targetcategory = self.category.get()
        self.notelist = [ (t,c) for (t,c) in self.notes if c == self.targetcategory ]
        if len(self.notelist) > 0:
            (t0, c0) = self.notelist[0]
            self.title.set(t0)
            self.note.set(self.notes[(t0,c0)])
            self.notelist.remove((t0, c0))
        else:
            tk.messagebox.showinfo("Error", "No matching notes found for that category")

    def next_note(self):
        print("pressed next")
        if len(self.notelist) > 0:
            (t0, c0) = self.notelist[0]
            self.title.set(t0)
            self.note.set(self.notes[(t0,c0)])
            self.notelist.remove((t0, c0))
        else:
            tk.messagebox.showinfo("Error", "No more matching notes found for that category")



root = tk.Tk()
app = NoteApp(root)
app.notes = dict()
app.notelist = list()
app.dirty_bit = 0
root.mainloop()
```

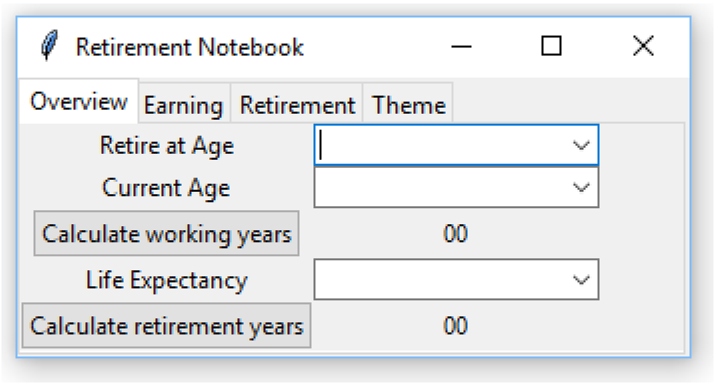
That's better code, but we still have not intercepted the close X on the window's title bar. That requires Tcl working with the operating system. Being an advanced topic it will be handled in a future chapter.

6.1 Notebook

The notebook widget provides an easily comprehended interface for the display of text and images grouped by page. That model closely follows the prototypical example of a paper notebook. A series of tabs at the top of the window permit easy movement from page to page. Notebooks could easily be applied to the development of user manuals and repair guides.

Since notebooks can easily be nested the notebook widget aids the development of a system to categorize all the pertinent information within a top-down tree structure. But, realistically Web pages have already conquered this territory, so the notebook will remain in limited use.

Notebooks can also be used to group portions of an interface when the underlying problem appears in stages. This example problem involves accumulating financial resources for retirement and then estimating how long those resources will last. Thus the problem can easily be split into two stages: an accumulation stage and a retirement stage. Obviously determining the year to begin retirement is an important concern and earns its own page in the notebook.



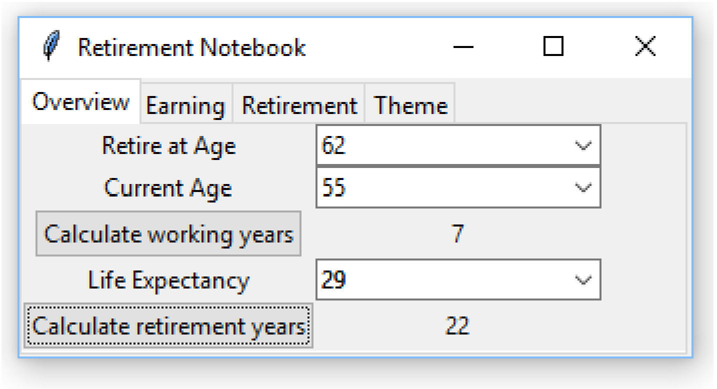
The user begins by selecting her age at retirement. Obviously this choice could be the result of the computations completed by this application, so the user is left in a quandary: what year to enter. Users may not be aware of the “what if” approach to solving a problem. It may have predated the computer, but the term became common beginning with Dan Bricklin and Bob Frankston's application named VisiCalc which appeared in 1979 for use on the Apple II computer. Succinctly a person would guess an age of retirement. Next, the user would run the application using that initial guess. After the application completed its computations and produced its results, the user would adjust the retirement age and rerun the application. Only after numerous executions of the application using a variety of retirement ages would the user begin to get appropriate results. Wikipedia at https://en.wikipedia.org/wiki/Sensitivity_analysis has a description of this approach written for readers with exceptional skills in reading and mathematics.

In most life situations the user knows her own age. If that is a problem then that problem is beyond the scope of this application. Next the user should the button labeled "Calculate working years." This application was implemented to demonstrate use of the notebook widget, so this application does practice safe computing. If the user forgets to press that button, then the program may later crash or produce erroneous results.

Next the user enters her longevity. During prior chapters you were exposed to the complexity of making that decision. Indeed entire books have been written on the topic of longevity. This example program asks the user to enter the result of that decision process.

Having two indeterminate variables on the same page makes this application especially challenging for the user. We cannot escape this complexity because that's life. Finally the user must press the button labeled "Calculate retirement years." Again failure to follow this sequence of steps will result in errors.

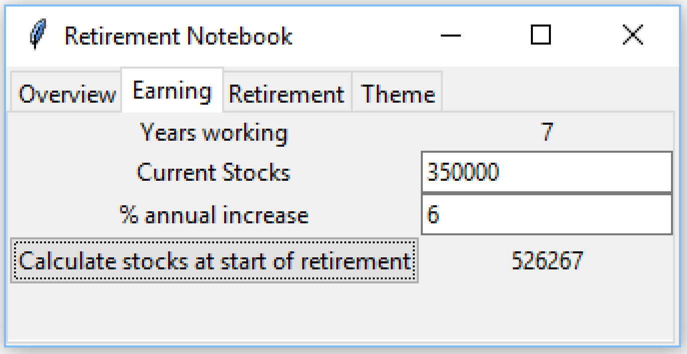
Let's assume the user is age 55, plans to retire at age 62, and social security predicts on average at age 84, so the longevity is 29 years. Thus she has only seven years to accumulate enough to for 22 years of retirement.



Retire at Age 62
Current Age 55
7
Life Expectancy 29
22

Now that the basic schedule for one's life has been selected, the user can advance to the next page of the notebook.

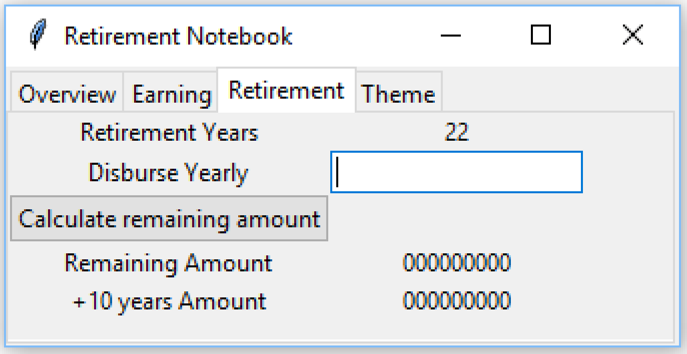
Notice the years remaining in her work career have been carried over to the earnings page. The user now enters the current amount of her retirement funds, which at $350,000 is considered above average. Next she must estimate future earnings. Of course Wall Street pundits cannot agree, but they do write books on that topic.


Retirement Notebook

Years working 7
Current Stocks 350000
% annual increase 6

526267

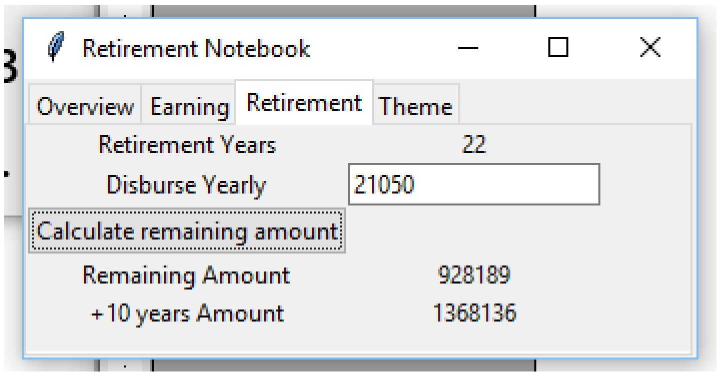
Pressing the "Calculate stocks at start of retirement" button resulted in $526,267 at the start of her retirement. Go to page three to see if this amount will last her for the rest of her life.


Retirement Notebook

Retirement Years 22
Disburse Yearly

Remaining Amount 000000000
+10 years Amount 000000000

Notice the years of retirement has carried over to this page. Next she must estimate her annual expenses during retirement. A common rule is named the 4% rule whereas at the start of her retirement she computes 4% of her retirement funds. Then she withdraws that amount annually adjusting it upwards for inflation. Hopefully she will receive social security, which averages approximate $1,100 per month. If she needs $3,000 monthly to barely survive, then the funds need to generate $1,900 per month. Four percent of her $526,267 would be $21,050 per year or $1,754 per month. Her social security of $1,100 together with that 4 percent withdrawal of $1,754 would total $2,854 per month for a challenging retirement.



Retirement Years 22
Disburse Yearly 21050
Remaining Amount 928189
+10 years Amount 1368136

Pressing the calculation button produced a prediction that her retirement funds could grow to $928,189 by the time she retired and $1,368,136 if .she should live ten years longer than her predicted longevity. Again this page is a what-if situation, so she could increase her withdrawal amount.

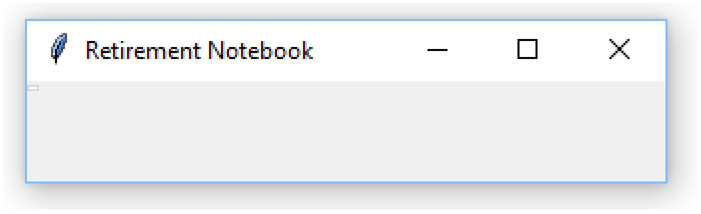
Certainly this application has three what-if variables for the user to manipulate. She could be running this applications with different values for a long time before she arrives at an understanding of her situation. She should consult with an expert on investing for retirement.

That's the application. Now let's look at the code.

First we add the notebook widget to the main frame.

```
from tkinter import ttk
import tkinter as tk

def app():
    root = tk.Tk()
    root.title("Retirement Notebook")

    nb = ttk.Notebook(root)

    nb.grid(row=1,column=1)

    root.mainloop()

if __name__ == "__main__":
    app()
```

Of course that produces no display. Check the window below.

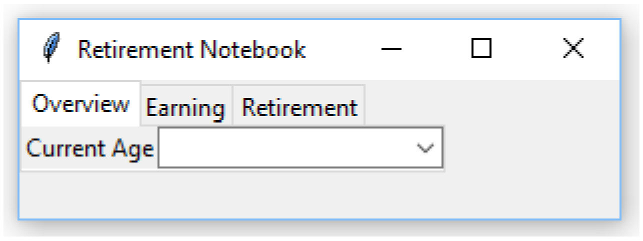


You can barely see in the top left corner a small symbol.

Next, we add three pages to that notebook.

```
from tkinter import ttk
import tkinter as tk

def app():
    root = tk.Tk()
    root.title("Retirement Notebook")

    nb = ttk.Notebook(root)

    # adding Frames as pages for the ttk.Notebook
    # first page, which would get widgets gridded into it
    overview_page1 = ttk.Frame(nb)
    current_age_lbl = ttk.Label(overview_page1, text="Current Age")
    current_age_lbl.grid(row=2, column=1)
    current_age_text = tk.StringVar()
    current_age_combo = ttk.Combobox(overview_page1, textvariable=current_age_text)
    current_age_combo.grid(row=2, column=2)

    # second page
    work_years_page2 = ttk.Frame(nb)

    # third page
    retired_years_page3 = ttk.Frame(nb)

    nb.add(overview_page1, text='Overview')
    nb.add(work_years_page2, text='Earning')
    nb.add(retired_years_page3, text='Retirement')

    nb.grid(row=1,column=1)

    root.mainloop()

if __name__ == "__main__":
    app()
```

Only the overview page has any content. It's shown below. No python code supports that window.



At least we can move around those nearly empty pages. Next we populate those pages with labels, fields, and buttons. And, we add the supporting functions.

```
from tkinter import ttk
import tkinter as tk

def RetirementApp():

    rate = 0.0
    remaining_years_work = 0
    retired_years = 0
    retire_with_amt = 0

    #Page 1: Overview Page
    def calc_working_years():
        global remaining_years_work
        print('retirement age :' + retire_age_text.get() + ":")
        print('current age :' + current_age_text.get() + ":")
        remaining_years_work = int(retire_age_text.get()) - int(current_age_text.get())
        #page 1: Overview
        working_years_text["text"] = str(remaining_years_work)
        #page 2: Earning
        years_to_work_text["text"] = str(remaining_years_work)

    def calc_retired_years():
        global retired_years
        retired_years = int(life_expectancy_text.get()) - (int(retire_age_text.get()) -
int(current_age_text.get()))
        #page 1: Overview
        retired_years_text["text"] = str(retired_years)
        #page 3: Retirement
        retirement_years_text_lbl["text"] = str(retired_years)

    #page 2: Earnings Page
    def calc_stocks_at_start_retired_years():
        global rate
        global retire_with_amt
        remaining_years_work = int(retire_age_text.get()) - int(current_age_text.get())
        retired_years = int(life_expectancy_text.get()) - (int(retire_age_text.get()) -
int(current_age_text.get()))
        retire_with_amt = int(current_amt_text.get())
        years_life_expectancy = int(life_expectancy_text.get())
        rate = float(annual_rate_text.get())
        rate = 1.0 + (rate / 100)
        #print("rate: " + str(rate))
        for yr in range(remaining_years_work):
            retire_with_amt = (int)(retire_with_amt * (rate))
            #print("amt: " + str(retire_with_amt))
        stocks_at_start_retired_years_text["text"] = str(retire_with_amt)

    #page 3: Retirement
    def calc_remaining_amt():
        global rate
        global retired_years
        global retire_with_amt
        disburse = int(disburse_text.get())
        for yr in range(retired_years):
            retire_with_amt -= disburse
            retire_with_amt = (int)(retire_with_amt * (rate))
            #print("amt: " + str(retire_with_amt))
        remainder_lbl_amt["text"] = str(retire_with_amt)
        for yr in range(10):
            retire_with_amt -= disburse
            retire_with_amt = (int)(retire_with_amt * (rate))
            #print("amt: " + str(retire_with_amt))
        remainder_10more_amt_lbl["text"] = str(retire_with_amt)


    root = tk.Tk()
    root.title("Retirement Notebook")

    nb = ttk.Notebook(root)

    # adding Frames as pages for the ttk.Notebook
    # first page: Overview
    overview_page1 = ttk.Frame(nb)
    retire_age_lbl = ttk.Label(overview_page1, text="Retire at Age")
    retire_age_lbl.grid(row=2, column=1)
    retire_age_text = tk.StringVar()
    year_options = tuple([str(n) for n in range(1,120)])
    retire_age_combo = ttk.Combobox(overview_page1, values=year_options,
textvariable=retire_age_text)
    retire_age_combo.grid(row=2, column=2)

    current_age_lbl = ttk.Label(overview_page1, text="Current Age")
    current_age_lbl.grid(row=3, column=1)
    current_age_text = tk.StringVar()
    year_options = tuple([str(n) for n in range(1,120)])
    current_age_combo = ttk.Combobox(overview_page1, values=year_options,
textvariable=current_age_text)
    current_age_combo.grid(row=3, column=2)

##    calc_working_years_btn = ttk.Button(overview_page1, text="Calculate working years",
style='C.TButton', command=calc_working_years)
    calc_working_years_btn = ttk.Button(overview_page1, text="Calculate working years",
command=calc_working_years)
    calc_working_years_btn.grid(row=4, column=1)
    working_years_text = ttk.Label(overview_page1, text="00")
    working_years_text.grid(row=4, column=2)

    life_expectancy_lbl = ttk.Label(overview_page1, text="Life Expectancy")
    life_expectancy_lbl.grid(row=5, column=1)
    life_expectancy_text = tk.StringVar()
    life_expectancy_combo = ttk.Combobox(overview_page1, values=year_options,
textvariable=life_expectancy_text)
    life_expectancy_combo.grid(row=5, column=2)

##    calc_working_years_btn = ttk.Button(overview_page1, text="Calculate retirement years",
style='C.TButton', command=calc_retired_years)
    calc_working_years_btn = ttk.Button(overview_page1, text="Calculate retirement years",
command=calc_retired_years)
    calc_working_years_btn.grid(row=6, column=1)
    retired_years_text = ttk.Label(overview_page1, text="00")
    retired_years_text.grid(row=6, column=2)

    # second page: Earnings
    work_years_page2 = ttk.Frame(nb)
    years_to_work_lbl = ttk.Label(work_years_page2, text="Years working")
    years_to_work_lbl.grid(row=1, column=1)
    years_to_work_text = ttk.Label(work_years_page2, text="00")
    years_to_work_text.grid(row=1, column=2)

    current_stocks_lbl = ttk.Label(work_years_page2, text="Current Stocks")
    current_stocks_lbl.grid(row=2, column=1)
    current_amt_text = tk.StringVar()
    current_stocks_entry = ttk.Entry(work_years_page2, textvariable=current_amt_text)
    current_stocks_entry.grid(row=2, column = 2)
    annual_stocks_increase_lbl = ttk.Label(work_years_page2, text="% annual increase")
    annual_stocks_increase_lbl.grid(row=3, column=1)
    annual_rate_text = tk.StringVar()
    annual_stocks_increase_entry = ttk.Entry(work_years_page2, textvariable=annual_rate_text)
    annual_stocks_increase_entry.grid(row=3, column = 2)
##    calc_stocks_at_retirement_btn = ttk.Button(work_years_page2, text="Calculate stocks at
start of retirement",
##                                      style='C.TButton',
command=calc_stocks_at_start_retired_years)
    calc_stocks_at_retirement_btn = ttk.Button(work_years_page2, text="Calculate stocks at
start of retirement", command=calc_stocks_at_start_retired_years)
    calc_stocks_at_retirement_btn.grid(row=4, column=1)
    stocks_at_start_retired_years_text = ttk.Label(work_years_page2, text="00")
    stocks_at_start_retired_years_text.grid(row=4, column=2)


    # third page: Retirement
    retired_years_page3 = ttk.Frame(nb)
    retirement_years_lbl = ttk.Label(retired_years_page3, text="Retirement Years")
    retirement_years_lbl.grid(row=2, column=1)
    retirement_years_text_lbl = ttk.Label(retired_years_page3)
    retirement_years_text_lbl.grid(row=2, column = 2)
    disburse_yearly_lbl = ttk.Label(retired_years_page3, text="Disburse Yearly")
    disburse_yearly_lbl.grid(row=3, column=1)
    disburse_text = tk.StringVar()
    disburse_yearly_entry = ttk.Entry(retired_years_page3, textvariable=disburse_text)
    disburse_yearly_entry.grid(row=3, column = 2)
##    calc_remaining_btn = ttk.Button(retired_years_page3, text="Calculate remaining amount",
style='C.TButton', command=calc_remaining_amt)
    calc_remaining_btn = ttk.Button(retired_years_page3, text="Calculate remaining amount",
command=calc_remaining_amt)
    calc_remaining_btn.grid(row=4, column=1)
    remainder_lbl = ttk.Label(retired_years_page3, text="Remaining Amount")
    remainder_lbl.grid(row=5, column=1)
    remainder_lbl_amt = ttk.Label(retired_years_page3, text="000000000")
    remainder_lbl_amt.grid(row=5, column=2)
    remainder_10more_lbl = ttk.Label(retired_years_page3, text="+10 years Amount")
    remainder_10more_lbl.grid(row=6, column=1)
    remainder_10more_amt_lbl = ttk.Label(retired_years_page3, text="000000000")
    remainder_10more_amt_lbl.grid(row=6, column=2)

    nb.add(overview_page1, text='Overview')
    nb.add(work_years_page2, text='Earning')
    nb.add(retired_years_page3, text='Retirement')

    nb.grid(row=1,column=1)

    root.mainloop()

if __name__ == "__main__":
    RetirementApp()
```

Although lengthy the program is straight forward and easy to comprehend. Notice how the variables are stored, so some information can be shared over several pages. Basically the variables are defined at the top of the function and sub-functions use the global parameter to access them. Remember to use the tk StringVar to access data from the ttk widgets. This avoids timing issues.

6.2 Styles and themes

Since the first chapter none of the example programs have styled any widgets. The Tk docs at http://www.tkdocs.com/tutorial/styles.html contain a clearly written introduction to styles and themes. Going beyond that introduction means studying Tcl/Tk sources. That is clearly beyond the scope of this manual.

In chapter one we saw how our python program could change the value of an attribute of a widget and thus changes its appearance. That approach to programming is too difficult to comprehend and debug. With the move to Tcl/Tk 8.5 our python programs should follow a more structured approach requiring more preparation and planning. A style simply means to change the appearance of a widget. A theme is a collection of such styles impacting all the available widgets. The tkdocs explain how to get a list of the currently available themes. The theme_names methods of a style object returns that list.

```
    >>> s = ttk.Style()
  >>> s.theme_names()
  ('aqua', 'step', 'clam', 'alt', 'default', 'classic')
```

To use another theme you simply use the following command.

```
    s.theme_use('themename')
```

Put together that becomes the heart of page four of our notebook. You can now change the theme as shown here.

During the initialization part of the application object we run the following code to get an array containing the names of the available themes.

```
  self.style = ttk.Style()
  available_themes = self.style.theme_names()
```

Then for page four we use a combobox loaded with that array. The button then has its command parameter set to the following function.

```
  set_theme_page4 = ttk.Frame(nb)
# create a Combobox with themes to choose from
  self.combo = ttk.Combobox(set_theme_page4, values=available_themes)
  self.combo.grid(row=1, column=1)
# make the Enter key change the style
  self.combo.bind('<Return>', self.change_style)
# make a Button to change the style
```

```
        button = ttk.Button(set_theme_page4, text='OK')
        button['command'] = self.change_style
        button.grid(row=2, column=1)
```

This function changes the theme being used.

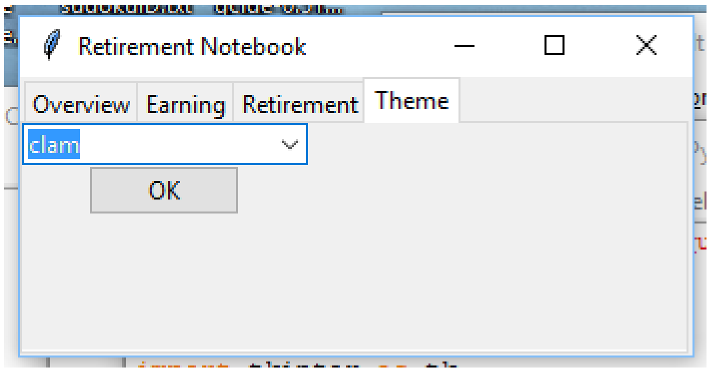
```
    def change_style(self, event=None):
        """set the Style to the content of the Combobox"""
        content = self.combo.get()
        try:
            self.style.theme_use(content)
        except tkinter.TclError as err:
            messagebox.showerror('Error', err)
        else:
            self.root.title(content)
```

The complete program is shown below.

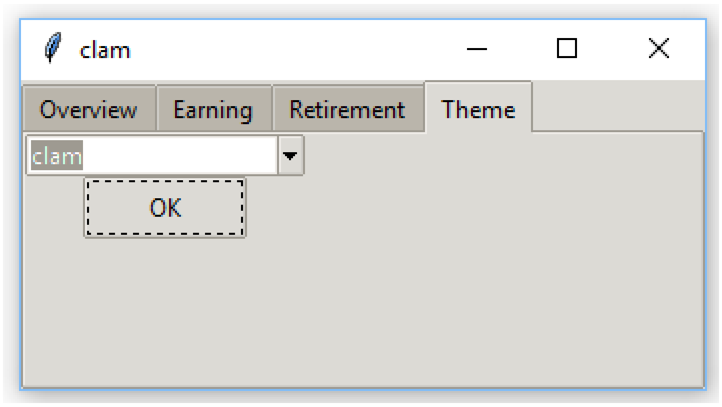
```
import random
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

class RetirementApp(object):

    #Page 1: Overview Page
    def calc_working_years(self):
        global remaining_years_work
##          print('retirement age :' + retire_age_text.get() + ":")
##          print('current age :' + current_age_text.get() + ":")
        self.remaining_years_work = int(self.retire_age_text.get()) - int(self.current_age_text.get())
        #page 1: Overview
        self.working_years_text["text"] = str(self.remaining_years_work)
        #page 2: Earning
        self.years_to_work_text["text"] = str(self.remaining_years_work)

    def calc_retired_years(self):
        global retired_years
        retired_years = int(self.life_expectancy_text.get()) - (int(self.retire_age_text.get()) -
int(self.current_age_text.get()))
        #page 1: Overview
        self.retired_years_text["text"] = str(retired_years)
        #page 3: Retirement
        self.retirement_years_text_lbl["text"] = str(retired_years)

    #page 2: Earnings Page
    def calc_stocks_at_start_retired_years(self):
        global rate
        global retire_with_amt
        remaining_years_work = int(self.retire_age_text.get()) - int(self.current_age_text.get())
        retired_years = int(self.life_expectancy_text.get()) - (int(self.retire_age_text.get()) -
int(self.current_age_text.get()))
        retire_with_amt = int(self.current_amt_text.get())
        years_life_expectancy = int(self.life_expectancy_text.get())
        rate = float(self.annual_rate_text.get())
        rate = 1.0 + (rate / 100)
##        print("rate: " + str(rate))
        for yr in range(remaining_years_work):
            retire_with_amt = (int)(retire_with_amt * (rate))
            #print("amt: " + str(retire_with_amt))
        self.stocks_at_start_retired_years_text["text"] = str(retire_with_amt)

    #page 3: Retirement
    def calc_remaining_amt(self):
        global rate
        global retired_years
        global retire_with_amt
        disburse = int(self.disburse_text.get())
        for yr in range(retired_years):
            retire_with_amt -= disburse
            retire_with_amt = (int)(retire_with_amt * (rate))
            #print("amt: " + str(retire_with_amt))
        self.remainder_lbl_amt["text"] = str(retire_with_amt)
        for yr in range(10):
            retire_with_amt -= disburse
            retire_with_amt = (int)(retire_with_amt * (rate))
            #print("amt: " + str(retire_with_amt))
        self.remainder_10more_amt_lbl["text"] = str(retire_with_amt)

    def change_style(self, event=None):
        """set the Style to the content of the Combobox"""
        content = self.combo.get()
        try:
            self.style.theme_use(content)
        except tkinter.TclError as err:
            messagebox.showerror('Error', err)
        else:
            self.root.title(content)



    def __init__(self):
        self.root = tk.Tk()
        self.style = ttk.Style()
        available_themes = self.style.theme_names()
##        random_theme = random.choice(available_themes)
##        self.style.theme_use(random_theme)
        self.root.title("Retirement Notebook")
        frm = ttk.Frame(self.root)
        frm.grid(row=0, column=0)

        rate = 0.0
        remaining_years_work = 0
        retired_years = 0
        retire_with_amt = 0
        nb = ttk.Notebook(frm)

        # adding Frames as pages for the ttk.Notebook
        # first page: Overview
        overview_page1 = ttk.Frame(nb)
        retire_age_lbl = ttk.Label(overview_page1, text="Retire at Age")
        retire_age_lbl.grid(row=2, column=1)
        self.retire_age_text = tk.StringVar()
        year_options = tuple([str(n) for n in range(1,120)])
        self.retire_age_combo = ttk.Combobox(overview_page1, values=year_options,
textvariable=self.retire_age_text)
        self.retire_age_combo.grid(row=2, column=2)

        current_age_lbl = ttk.Label(overview_page1, text="Current Age")
        current_age_lbl.grid(row=3, column=1)
        self.current_age_text = tk.StringVar()
        year_options = tuple([str(n) for n in range(1,120)])
        current_age_combo = ttk.Combobox(overview_page1, values=year_options,
textvariable=self.current_age_text)
        current_age_combo.grid(row=3, column=2)

    ##    calc_working_years_btn = ttk.Button(overview_page1, text="Calculate working years",
style='C.TButton', command=calc_working_years)
        calc_working_years_btn = ttk.Button(overview_page1, text="Calculate working years",
command=self.calc_working_years)
        calc_working_years_btn.grid(row=4, column=1)
        self.working_years_text = ttk.Label(overview_page1, text="00")
        self.working_years_text.grid(row=4, column=2)

        life_expectancy_lbl = ttk.Label(overview_page1, text="Life Expectancy")
        life_expectancy_lbl.grid(row=5, column=1)
        self.life_expectancy_text = tk.StringVar()
        life_expectancy_combo = ttk.Combobox(overview_page1, values=year_options,
textvariable=self.life_expectancy_text)
        life_expectancy_combo.grid(row=5, column=2)

    ##    calc_working_years_btn = ttk.Button(overview_page1, text="Calculate retirement
years", style='C.TButton', command=calc_retired_years)
        calc_working_years_btn = ttk.Button(overview_page1, text="Calculate retirement years",
command=self.calc_retired_years)
        calc_working_years_btn.grid(row=6, column=1)
        self.retired_years_text = ttk.Label(overview_page1, text="00")
        self.retired_years_text.grid(row=6, column=2)

        # second page: Earnings
        work_years_page2 = ttk.Frame(nb)
        years_to_work_lbl = ttk.Label(work_years_page2, text="Years working")
        years_to_work_lbl.grid(row=1, column=1)
        self.years_to_work_text = ttk.Label(work_years_page2, text="00")
        self.years_to_work_text.grid(row=1, column=2)

        current_stocks_lbl = ttk.Label(work_years_page2, text="Current Stocks")
        current_stocks_lbl.grid(row=2, column=1)
        self.current_amt_text = tk.StringVar()
        current_stocks_entry = ttk.Entry(work_years_page2, textvariable=self.current_amt_text)
        current_stocks_entry.grid(row=2, column = 2)
        annual_stocks_increase_lbl = ttk.Label(work_years_page2, text="% annual increase")
        annual_stocks_increase_lbl.grid(row=3, column=1)
        self.annual_rate_text = tk.StringVar()
        annual_stocks_increase_entry = ttk.Entry(work_years_page2,
textvariable=self.annual_rate_text)
        annual_stocks_increase_entry.grid(row=3, column = 2)
    ##    calc_stocks_at_retirement_btn = ttk.Button(work_years_page2, text="Calculate stocks
at start of retirement",
    ##                                             style='C.TButton',
command=calc_stocks_at_start_retired_years)
        calc_stocks_at_retirement_btn = ttk.Button(work_years_page2, text="Calculate stocks at
start of retirement",
                                             command=self.calc_stocks_at_start_retired_years)
        calc_stocks_at_retirement_btn.grid(row=4, column=1)
        self.stocks_at_start_retired_years_text = ttk.Label(work_years_page2, text="00")
        self.stocks_at_start_retired_years_text.grid(row=4, column=2)


        # third page: Retirement
        retired_years_page3 = ttk.Frame(nb)
        retirement_years_lbl = ttk.Label(retired_years_page3, text="Retirement Years")
        retirement_years_lbl.grid(row=2, column=1)
        self.retirement_years_text_lbl = ttk.Label(retired_years_page3)
        self.retirement_years_text_lbl.grid(row=2, column = 2)
        disburse_yearly_lbl = ttk.Label(retired_years_page3, text="Disburse Yearly")
        disburse_yearly_lbl.grid(row=3, column=1)
        self.disburse_text = tk.StringVar()
        disburse_yearly_entry = ttk.Entry(retired_years_page3, textvariable=self.disburse_text)
        disburse_yearly_entry.grid(row=3, column = 2)
    ##    calc_remaining_btn = ttk.Button(retired_years_page3, text="Calculate remaining
amount", style='C.TButton', command=calc_remaining_amt)
        calc_remaining_btn = ttk.Button(retired_years_page3, text="Calculate remaining amount",
command=self.calc_remaining_amt)
        calc_remaining_btn.grid(row=4, column=1)
        remainder_lbl = ttk.Label(retired_years_page3, text="Remaining Amount")
```

```
            remainder_lbl.grid(row=5, column=1)
            self.remainder_lbl_amt = ttk.Label(retired_years_page3, text="000000000")
            self.remainder_lbl_amt.grid(row=5, column=2)
            remainder_10more_lbl = ttk.Label(retired_years_page3, text="+10 years Amount")
            remainder_10more_lbl.grid(row=6, column=1)
            self.remainder_10more_amt_lbl = ttk.Label(retired_years_page3, text="000000000")
            self.remainder_10more_amt_lbl.grid(row=6, column=2)

            set_theme_page4 = ttk.Frame(nb)
        # create a Combobox with themes to choose from
            self.combo = ttk.Combobox(set_theme_page4, values=available_themes)
            self.combo.grid(row=1, column=1)
        # make the Enter key change the style
            self.combo.bind('<Return>', self.change_style)
        # make a Button to change the style
            button = ttk.Button(set_theme_page4, text='OK')
            button['command'] = self.change_style
            button.grid(row=2, column=1)

            nb.add(overview_page1, text='Overview')
            nb.add(work_years_page2, text='Earning')
            nb.add(retired_years_page3, text='Retirement')


            nb.add(set_theme_page4, text='Theme')

            nb.grid(row=1,column=1)

    app = RetirementApp()
    app.root.mainloop()
```

That program produces this window.



The window then changes its appearance as shown below.



The chosen theme in the combobox is clam.

So changing an application's theme is easy.

6.1 Text widget

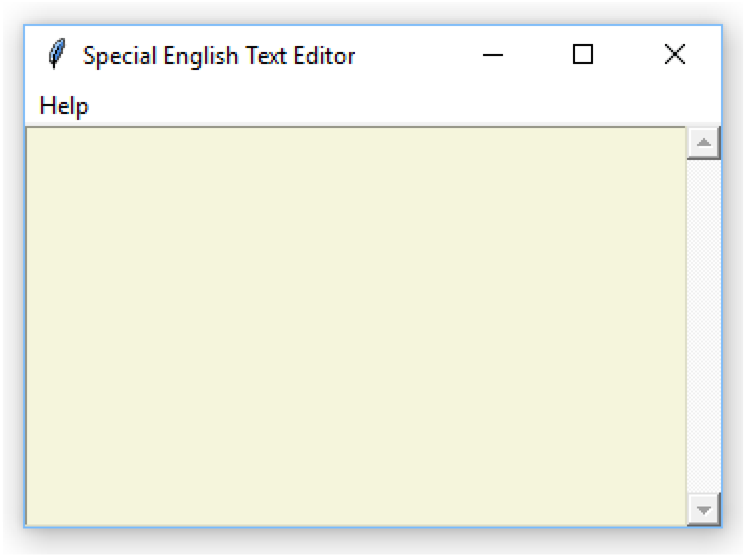
The text widget provides a multi-line window for manipulating text. It can also display images, search for text, and update text. Others have used this widget to build an IDE for python programming and have developed Web browsers. In this chapter we will use the text widget as the main part of text editor with the special twist that we will support writing in a tiny core of English. Wikipedia at https://en.wikipedia.org/wiki/Basic_English describes how Charles Ogden created Basic English, a severely limited dictionary of only 850 of the most basic words in the English language. It inspired some success in teaching English especially in foreign countries by using limited lists of English words. Voice of America produces a special edition of its publications using Special English which is a list of 1500 English words. Another severely limited subset of English named Simplified Technical English is promoted for writing technical manuals.

The first draft of our python program adds a text widget to a window.

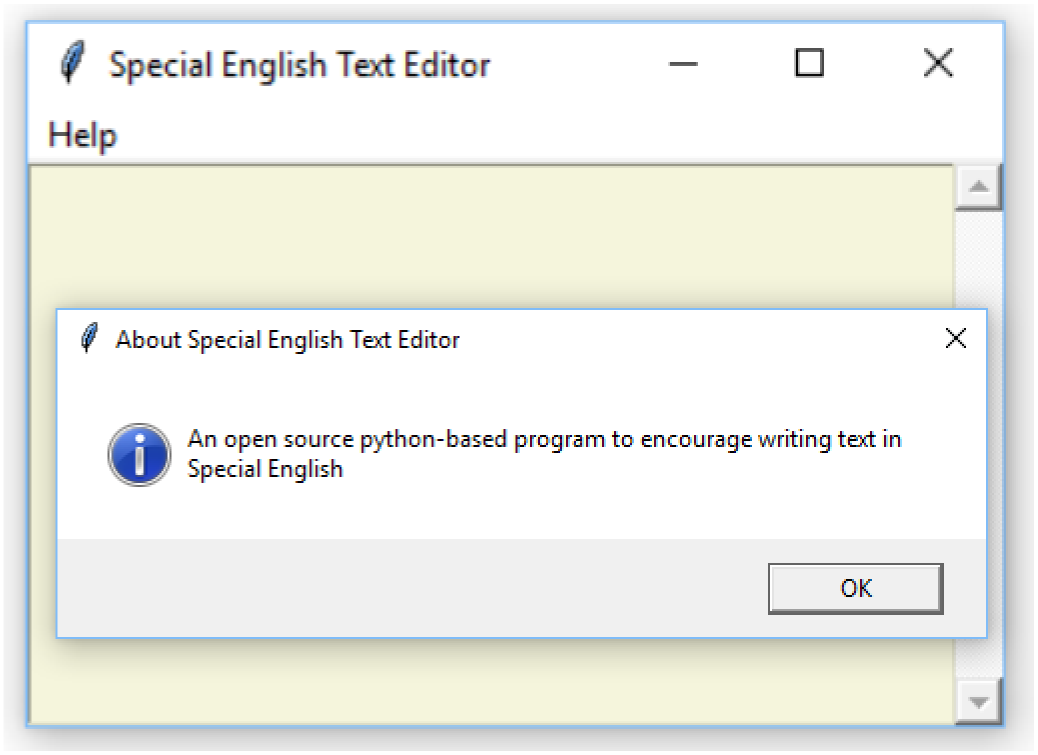
```
import tkinter as tk
import tkinter.scrolledtext as tkst

main_window = tk.Tk(className="Special English Text Editor")
main_window.title('Special English Text Editor')
main_window.geometry('300x200')
edit_space = tkst.ScrolledText(
   master = main_window,
   wrap = 'word', # wrap text at full words only
   undo = tk.TRUE,
   #width = 25, # characters
   #height = 10, # text lines
   bg='beige'  # background color of edit area
)

def about_desc():
   tk.messagebox.showinfo("About Special English Text Editor",
      "An open source python-based program to encourage writing text in Special English"
      )
#Create menus
main_menu = tk.Menu(main_window)
main_window.config(menu=main_menu)
```

```
help_menu = tk.Menu(main_menu)
main_menu.add_cascade(label="Help", menu=help_menu)
help_menu.add_command(label="About...", command=about_desc)
# end of menu creation
edit_space.pack(fill = tk.BOTH, expand = 1)

main_window.mainloop()
```

This program uses a scrolled text widget, which is simply a text widget with a scroll bar. That scrolled text widget reduces the work required to produces this GUI.



The text widget fills the window. It has a beige background color and supports our typing multi-line comments. Yes we face a major effort to support this widget. The menu has a single option to display the message shown below.


Special English Text Editor
Help
About Special English Text Editor

An open source python-based program to encourage writing text in Special English

The window's command of geometry('300x200') sets the size of the window as 300 pixels by 200 pixels. Tk docs provides a clear description of the window manipulation options at http://www.tkdocs.com/tutorial/windows.html. An example of the full command would be window.geometry('300x200-5+40') where the window is 300 pixels wide, 200 pixels high, 5 pixels from the left edge of the window, and 40 pixels from the top of the window. This reference to the tk docs should again encourage the reader to use the tk docs. Do not rely solely on python sources.

The widget_name.winfo_reqheight() method returns the height of the widget. Likewise widget_name.winfo_reqwidth() returns the width of the widget. Be aware that the geometry manager may change those values as the GUI is being stretched by the user. You can write a function to place the widget in the center of the window and specify how much of that window the widget will occupy.

Next we will create the file menu and add that file menu to the menu bar.

```
file_menu = Menu(main_menu)
main_menu.add_cascade(
   label="File",
   menu=file_menu)
```

The following python code adds the open file option to the file menu.

```
file_menu.add_command(
        label="Open",
        accelerator='Ctrl+O',
        compound=LEFT,
        image=open_file_icon,
        underline = 0,
        command=open_file
        )
```

Notice the accelerator option which connects the control and letter O keys with this menu option. The compound option allows both an image and the label to be displayed together as the menu option. The image option has the name of the variable for the icon.

```
    # file icons
    open_file_icon = PhotoImage(file='icons/open_file.gif')
```

So that icon is a GIF format image with the file name of open_file.gif, which is stored in the folder named icons. The underline option has the zero-based address of the letter to be underlined. For this example that means the leading letter O of “Open” will be underlined because that is the accelerator key for this file menu option. Check the directory you downloaded when you installed python 3.6. Those icons should be there.

Next we need the code to open the text file.

```
    def open_file(event=None):
        global file_name
        dirty_edit_space = False
        file_name = filedialog.askopenfilename(defaultextension=".txt",filetypes=[("All
    Files","*.*"),("Text Documents","*.txt")])
        if file_name == "": # If no file chosen.
            file_name = None # Absence of file.
        else:
            main_window.title(os.path.basename(file_name)) # Returning the basename of 'file'
            edit_space.delete(1.0,END)
            fh = open(file_name,"r")
            edit_space.insert(1.0,fh.read())
            fh.close()
```

Notice the file name is a global because other functions such as the save function will need to know that file's name. Next is the dirty bit which is named dirty_edit_space. The dirty bit indicates whether or not the edit space has been saved since the last changes were made to it. This function opens a text file and copies its contents to the edit space thus eliminating any prior editing work. Thus the dirty bit should be set to false.

A function is needed to set that dirty bit.

```
def on_key_press():
    global dirty_edit_space
    dirty_edit_space = True
```

And that function is called when any key is pressed. We bind the event associated with pressing any key to the on_key_press function by using the bind method for the edit_space together with the Tcl name of <Key>.

```
edit_space.bind('<Key>', on_key_press)
```

Similar functions have been written for the save menu options. The python program is shown below.

```
import tkinter as tk
import tkinter.scrolledtext as tkst

main_window = tk.Tk(className="Special English Text Editor")
main_window.title('Special English Text Editor')
main_window.geometry('300x200')
edit_space = tkst.ScrolledText(
    master = main_window,
    wrap = 'word', # wrap text at full words only
    undo = tk.TRUE,
    #width = 25, # characters
    #height = 10, # text lines
    bg='beige'  # background color of edit area
)

file_name = 'placeholder.txt'
dirty_edit_space = False

def on_key_press(e):
    global dirty_edit_space
    dirty_edit_space = True
    print("dirty bit set True")


# file icons
open_file_icon = tk.PhotoImage(file='icons/open_file.gif')
save_file_icon = tk.PhotoImage(file='icons/Save.gif')

# edit icons
undo_icon = tk.PhotoImage(file='icons/Undo.gif')
cut_icon = tk.PhotoImage(file='icons/Cut.gif')
copy_icon = tk.PhotoImage(file='icons/Copy.gif')
pasteicon = tk.PhotoImage(file='icons/Paste.gif')
```

```
def about_desc():
    tk.messagebox.showinfo("About Special English Text Editor",
        "An open source python-based program to encourage writing text in Special English"
        )

def open_file(event=None):
    global file_name
    dirty_edit_space = False
    file_name = filedialog.askopenfilename(defaultextension=".txt",filetypes=[("All
Files","*.*"),("Text Documents","*.txt")])
    if file_name == "": # If no file chosen.
        file_name = None # Absence of file.
    else:
        main_window.title(os.path.basename(file_name)) # Returning the basename of 'file'
        edit_space.delete(1.0,END)
        fh = open(file_name,"r")
        edit_space.insert(1.0,fh.read())
        fh.close()

# save file function
def save_file():
    global file_name
    global dirty_edit_space
    try:
        f = open(file_name, 'w')
        content = edit_space.get(1.0, 'end')
        f.write(content)
        f.close()
        dirty_edit_space = False
    except:
        file_name = None

def save_as():
    global file_name
    global dirty_edit_space
    file_name = filedialog.asksaveasfilename()
    try:
        f = open(file_name, 'w')
        content = edit_space.get(1.0, 'end')
        f.write(content)
        f.close()
        dirty_edit_space = False
    except:
        file_name = None

def exit_main_window():
    global dirty_edit_space
    if dirty_edit_space:
        isOK = tk.messagebox.askokcancel("Go ahead w/o saving new edits?", "OK?")
        if isOK:
            main_window.destroy()
            dirty_edit_space = False
    else:
         main_window.destroy()

#Create menus
main_menu = tk.Menu(main_window)
main_window.config(menu=main_menu)

file_menu = tk.Menu(main_menu)
main_menu.add_cascade(
    label="File",
    menu=file_menu)
file_menu.add_command(
    label="Open",
    accelerator='Ctrl+O',
    compound=tk.LEFT,
    image=open_file_icon,
    underline = 0,
    command=open_file
    )

file_menu.add_command(
    label="Save",
    accelerator='Ctrl+S',
    compound=tk.LEFT,
    image=save_file_icon,
    underline=0,
    command=save_file
    )
file_menu.add_command(label="Save As", command = save_as)
file_menu.add_separator()
file_menu.add_command(
    label="Exit",
    command = exit_main_window
    )

help_menu = tk.Menu(main_menu)
main_menu.add_cascade(label="Help", menu=help_menu)
help_menu.add_command(label="About...", command=about_desc)
# end of menu creation
edit_space.pack(fill = tk.BOTH, expand = 1)

edit_space.bind('<Control-O>', open_file)
edit_space.bind('<Control-o>', open_file)
edit_space.bind('<Control-S>', save_file)
edit_space.bind('<Control-s>', save_file)
```

```
#edit_space.bind('<Control-A>', select_all)
#edit_space.bind('<Control-a>', select_all)
#edit_space.bind('<Control-f>', on_find)
#edit_space.bind('<Control-F>', on_find)
edit_space.bind('<Key>', on_key_press)

main_window.mainloop()
```

For the next stage of this program we will add the edit menu and its options.

```
edit_menu = Menu(main_menu)
main_menu.add_cascade(label="Edit", menu=edit_menu)
edit_menu.add_command(
    label="Undo",
    compound=LEFT,
    image=undo_icon,
    accelerator='Ctrl+Z',
    command = undo)
edit_menu.add_separator()
edit_menu.add_command(
    label="Cut",
    compound = LEFT,
    image = cut_icon,
    accelerator='Ctrl+X',
    command = cut)
edit_menu.add_command(
    label = "Copy",
    compound = LEFT,
    image = copy_icon,
    accelerator='Ctrl+C',
    command = copy)
edit_menu.add_command(
    label="Paste",
    compound=LEFT,
    image=pasteicon,
    accelerator='Ctrl+V',
    command = paste)
edit_menu.add_separator()
edit_menu.add_command(
    label = "Find",
    underline = 0,
    accelerator = 'Ctrl+F',
    command = on_find)
#edit_menu.add_command(label="Select All", underline=7, accelerator='Ctrl+A',
command=select_all)
edit_menu.add_command(
    label="Words",
    #underline= 0,
    #accelerator='Ctrl+W',
    command = check_words)
edit_menu.add_command(
    label="Dictionary",
    #underline= 0,
    #accelerator='Ctrl+W',
    command = select_dictionaries)
```

Next we include the supporting functions for those edit commands.

```
    def undo(event = None):
        edit_space.event_generate("<<Undo>>") # <<Undo>> is TCL command, not python

    def cut(event = None):
        edit_space.event_generate("<<Cut>>")  # <<Cut>> is TCL command

    def copy(event = None):
        edit_space.event_generate("<<Copy>>") # <<Copy>> is TCL command

    def paste(event = None):
        edit_space.event_generate("<<Paste>>") # <<Paste>> is TCL command
```

So our functions generate an event for the text widget and call the matching Tcl command which is surrounded by << and >>. The text widget does reduce our work.

The next task is to implement a find command. This code adds the find command to the edit menu.

```
    edit_menu.add_command(
        label = "Find",
        underline = 0,
        accelerator = 'Ctrl+F',
        command = on_find)
```

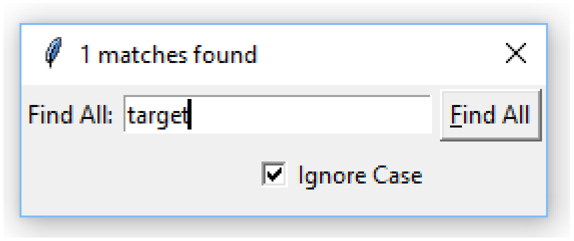
So, now we must write the on_find function.

```
    def on_find():
        find_window = Toplevel(main_window)
        find_window.title('Find')
        find_window.geometry('262x65+200+250')
        find_window.transient(main_window)
        find_window.bg='beige'  # background color of edit area
        Label(find_window, text="Find All:").grid(row=0, column=0, sticky='e')
        find_value = StringVar()
        entry_field = Entry(find_window, width=25, textvariable = find_value)
        entry_field.grid(row=0, column=1, padx=2, pady=2, sticky='we')
        entry_field.focus_set()
        ignore_case = IntVar()
        Checkbutton(find_window, text='Ignore Case', variable=ignore_case).grid(row=1, column=1,
    sticky='e', padx=2, pady=2)
        Button(
            find_window,
            text = "Find All",
            underline = 0,
            command = lambda: search_for(find_value.get(), ignore_case.get(), edit_space,
    find_window, entry_field)).grid(row = 0, column = 2, sticky = 'e' + 'w', padx = 2, pady = 2)
        def close_search():
            edit_space.tag_remove('match', '1.0', END)
            find_window.destroy()
        find_window.protocol('WM_DELETE_WINDOW', close_search) # override close button
```

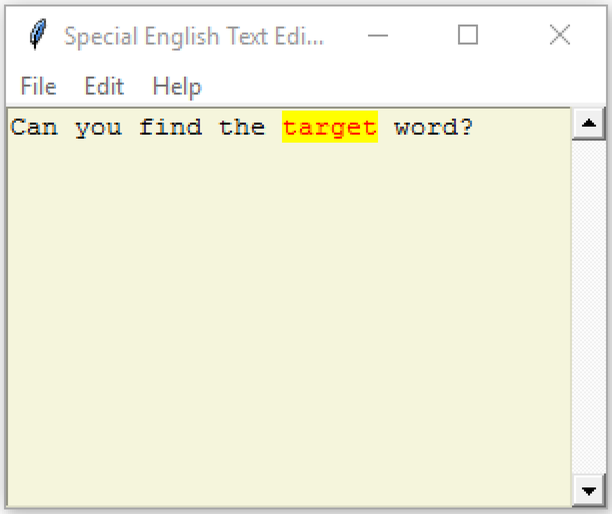
That function creates a new top level window. The user enter the word to be found and presses the find all button. Then the search_for lambda function is called with the search target as its parameter.

```
# find the needle in the haystack
def search_for(needle, ignore_case, text_haystack, find_window, entry_field):
    edit_space.tag_remove('match', '1.0', tk.END)
    count =0
    if needle:
        pos = '1.0'
        while True:
            pos = text_haystack.search(needle, pos, nocase = ignore_case, stopindex = tk.END)
            if not pos: break
            lastpos = '%s+%dc' % (pos, len(needle))
            text_haystack.tag_add('match', pos, lastpos)
            count += 1
            pos = lastpos
        text_haystack.tag_config('match', foreground='red', background='yellow')
    entry_field.focus_set()
    find_window.title('%d matches found' %count)
```

This function removes all the match tags from the text widget beginning from row 1 column 0 to the end. Then it calls the text widget's search method. It requests a search for the target word beginning from the first row through the end of the edit space. Because that call is in a while loop this function will continue searching for all matches until the end is reached. Every time the text widget's search method finds a match it tags that text with a match tag. Finally it works through those match tags changing the color to red and background color to yellow.



We are searching for the word: target. Notice how it's color changes to red.



```
Can you find the target word?
```

It should be clear now how much the text widget makes our work easier. Let's look over the entire program again to grasp the ease of implementing a word processor.

```
import tkinter as tk
import tkinter.scrolledtext as tkst

main_window = tk.Tk(className="Special English Text Editor")
main_window.title('Special English Text Editor')
main_window.geometry('300x200')
edit_space = tkst.ScrolledText(
    master = main_window,
    wrap = 'word', # wrap text at full words only
    undo = tk.TRUE,
    #width = 25, # characters
    #height = 10, # text lines
    bg='beige'  # background color of edit area
)

file_name = 'placeholder.txt'
dirty_edit_space = False
```

```
def undo(event = None):
    edit_space.event_generate("<<Undo>>") # <<Undo>> is TCL command, not python

def cut(event = None):
    edit_space.event_generate("<<Cut>>")  # <<Cut>> is TCL command

def copy(event = None):
    edit_space.event_generate("<<Copy>>") # <<Copy>> is TCL command

def paste(event = None):
    edit_space.event_generate("<<Paste>>") # <<Paste>> is TCL command

def on_key_press(e):
    global dirty_edit_space
    dirty_edit_space = True

# file icons
open_file_icon = tk.PhotoImage(file='icons/open_file.gif')
save_file_icon = tk.PhotoImage(file='icons/Save.gif')

# edit icons
undo_icon = tk.PhotoImage(file='icons/Undo.gif')
cut_icon = tk.PhotoImage(file='icons/Cut.gif')
copy_icon = tk.PhotoImage(file='icons/Copy.gif')
pasteicon = tk.PhotoImage(file='icons/Paste.gif')

def about_desc():
    tk.messagebox.showinfo("About Special English Text Editor",
        "An open source python-based program to encourage writing text in Special English"
        )

def open_file(event=None):
    global file_name
    dirty_edit_space = False
    file_name = filedialog.askopenfilename(defaultextension=".txt",filetypes=[("All
Files","*.*"),("Text Documents","*.txt")])
    if file_name == "": # If no file chosen.
        file_name = None # Absence of file.
    else:
        main_window.title(os.path.basename(file_name)) # Returning the basename of 'file'
        edit_space.delete(1.0,END)
        fh = open(file_name,"r")
        edit_space.insert(1.0,fh.read())
        fh.close()

# save file function
def save_file():
    global file_name
    global dirty_edit_space
    try:
        f = open(file_name, 'w')
        content = edit_space.get(1.0, 'end')
        f.write(content)
        f.close()
        dirty_edit_space = False
    except:
        file_name = None

def save_as():
    global file_name
    global dirty_edit_space
    file_name = filedialog.asksaveasfilename()
    try:
        f = open(file_name, 'w')
        content = edit_space.get(1.0, 'end')
        f.write(content)
        f.close()
        dirty_edit_space = False
    except:
        file_name = None

def exit_main_window():
    global dirty_edit_space
    if dirty_edit_space:
        isOK = tk.messagebox.askokcancel("Go ahead w/o saving new edits?", "OK?")
        if isOK:
            main_window.destroy()
            dirty_edit_space = False
    else:
         main_window.destroy()

def on_find():
    find_window = tk.Toplevel(main_window)
    find_window.title('Find')
    find_window.geometry('262x65+200+250')
    find_window.transient(main_window)
    find_window.bg='beige'  # background color of edit area
    tk.Label(find_window, text="Find All:").grid(row=0, column=0, sticky='e')
    find_value = tk.StringVar()
    entry_field = tk.Entry(find_window, width=25, textvariable = find_value)
    entry_field.grid(row=0, column=1, padx=2, pady=2, sticky='we')
    entry_field.focus_set()
    ignore_case = tk.IntVar()
    tk.Checkbutton(find_window, text='Ignore Case', variable=ignore_case).grid(row=1,
column=1, sticky='e', padx=2, pady=2)
    tk.Button(
        find_window,
        text = "Find All",
        underline = 0,
        command = lambda: search_for(find_value.get(), ignore_case.get(), edit_space,
find_window, entry_field)).grid(row = 0, column = 2, sticky = 'e' + 'w', padx = 2, pady = 2)
    def close_search():
        edit_space.tag_remove('match', '1.0', tk.END)
        find_window.destroy()
    find_window.protocol('WM_DELETE_WINDOW', close_search) # override close button

# find the needle in the haystack
def search_for(needle, ignore_case, text_haystack, find_window, entry_field):
    edit_space.tag_remove('match', '1.0', tk.END)
    count =0
    if needle:
        pos = '1.0'
        while True:
            pos = text_haystack.search(needle, pos, nocase = ignore_case, stopindex = tk.END)
            if not pos: break
            lastpos = '%s+%dc' % (pos, len(needle))
            text_haystack.tag_add('match', pos, lastpos)
            count += 1
            pos = lastpos
        text_haystack.tag_config('match', foreground='red', background='yellow')
    entry_field.focus_set()
    find_window.title('%d matches found' %count)


#Create menus
main_menu = tk.Menu(main_window)
main_window.config(menu=main_menu)

file_menu = tk.Menu(main_menu)
main_menu.add_cascade(
    label="File",
    menu=file_menu)
file_menu.add_command(
    label="Open",
    accelerator='Ctrl+O',
    compound=tk.LEFT,
    image=open_file_icon,
    underline = 0,
    command=open_file
    )

file_menu.add_command(
    label="Save",
    accelerator='Ctrl+S',
    compound=tk.LEFT,
    image=save_file_icon,
    underline=0,
    command=save_file
    )
file_menu.add_command(label="Save As", command = save_as)
file_menu.add_separator()
file_menu.add_command(
    label="Exit",
    command = exit_main_window
    )

edit_menu = tk.Menu(main_menu)
main_menu.add_cascade(label="Edit", menu=edit_menu)
edit_menu.add_command(
    label="Undo",
    compound=tk.LEFT,
    image=undo_icon,
    accelerator='Ctrl+Z',
    command = undo)
edit_menu.add_separator()
edit_menu.add_command(
    label="Cut",
    compound = tk.LEFT,
    image = cut_icon,
    accelerator='Ctrl+X',
    command = cut)
edit_menu.add_command(
    label = "Copy",
    compound = tk.LEFT,
    image = copy_icon,
    accelerator='Ctrl+C',
    command = copy)
edit_menu.add_command(
    label="Paste",
    compound=tk.LEFT,
    image=pasteicon,
    accelerator='Ctrl+V',
    command = paste)
edit_menu.add_separator()
edit_menu.add_command(
    label = "Find",
    underline = 0,
    accelerator = 'Ctrl+F',
    command = on_find)
#edit_menu.add_command(label="Select All", underline=7, accelerator='Ctrl+A',
command=select_all)


help_menu = tk.Menu(main_menu)
main_menu.add_cascade(label="Help", menu=help_menu)
help_menu.add_command(label="About...", command=about_desc)
```

```
# end of menu creation
edit_space.pack(fill = tk.BOTH, expand = 1)

edit_space.bind('<Control-O>', open_file)
edit_space.bind('<Control-o>', open_file)
edit_space.bind('<Control-S>', save_file)
edit_space.bind('<Control-s>', save_file)
edit_space.bind('<Control-f>', on_find)
edit_space.bind('<Control-F>', on_find)
edit_space.bind('<Key>', on_key_press)

main_window.mainloop()
```

Now to collect entries for our dictionary of acceptable words.

```
edit_menu.add_command(
   label="Words",
   #underline= 0,
   #accelerator='Ctrl+W',
   command = check_words)
edit_menu.add_command(
   label="Dictionary",
   #underline= 0,
   #accelerator='Ctrl+W',
   command = select_dictionaries)
```

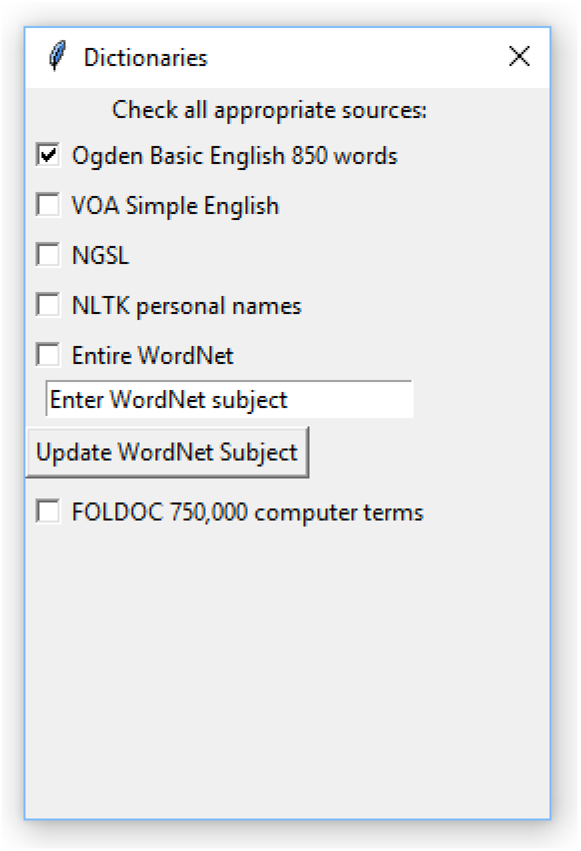
That adds options to the edit menu to select which dictionaries to gather words from and to check the edit space's contents against that master list.

The function named select_dictionaries permits the user to select Ogden's 850 word list, the Voice of America's simple Englist list, a list of names, the approximately 100,000 words in the WordNet collection, and FOLDOC dictionary of computing terms.

```
def select_dictionaries():
   global use_basic_english
   global use_voa_simple_english
   global voa_simple_english_words
   global names
   global use_nltk_names
   global nltk_names
   global use_wordnet
   global use_wordnet_subject
   global wn_subject
   global use_csv_dictionary
   global use_foldoc
   #global var1
   #global var2
```

```
    #global var3
    #global var4
    wn_subjects = {}
    dictionary_window = Toplevel(main_window)
    dictionary_window.title('Dictionaries')
    dictionary_window.geometry('262x365+200+250')
    dictionary_window.bg='beige'  # background color of edit area
    dictionary_window.transient(main_window)
    Label(dictionary_window, text="Check all appropriate sources:").grid(row=0, column=0,
sticky='e')
    var1 = IntVar()
    def update_basic_english():
        global use_basic_english
        nonlocal var1
        if var1.get() == 1:
            use_basic_english = True
            print(var1.get())
            print("using Basic English")
        else:
            use_basic_english = False
            print(var1.get())
            print("not using Basic English")
    chk_btn_basic = Checkbutton(dictionary_window, text="Ogden Basic English 850 words",
variable=var1, command=update_basic_english)
    chk_btn_basic.grid(row=1, sticky=W)
    chk_btn_basic.select()
    var2 = IntVar()
    def update_voa_simple_english():
        global use_voa_simple_english
        global voa_simple_english_words
        nonlocal var2
        if var2.get() == 1:
            use_voa_simple_english = True
            print(var2.get())
            print("loading VOA Simple English")
            voa_simple_english_words_corpus = PlaintextCorpusReader('./','voaSpecialEnglish.txt')
            file_ids = voa_simple_english_words_corpus.fileids()
            #print("VOA Simple English file info")
            #for f in file_ids:
                #print("file id: ", f)
            voa_simple_english_words = voa_simple_english_words_corpus.words() # returns list
of string
            length = len(voa_simple_english_words)
            print("VOA Special English has ", length, " words.")
        else:
            use_voa_simple_english = False
            print(var2.get())
            print("not using VOA Simple English")
    Checkbutton(dictionary_window, text="VOA Simple English", variable=var2,
command=update_voa_simple_english).grid(row=2, sticky=W)
    var2b = IntVar()
    def update_ngsl():
        global use_ngsl
        global ngsl_words
        nonlocal var2b
        if var2b.get() == 1:
            use_ngsl = True
            print(var2b.get())
            print("loading New General Service List")
            ngsl_words_corpus = PlaintextCorpusReader('./','ngsl_Headwords.txt')
            #file_ids = voa_simple_english_words_corpus.fileids()
            #print("VOA Simple English file info")
            #for f in file_ids:
                #print("file id: ", f)
            ngsl_words = ngsl_words_corpus.words() # returns list of string
            length = len(ngsl_words)
            print("NGSL New General Service List has ", length, " words.")
        else:
            use_ngsl = False
            print(var2b.get())
            print("not using NGSL New General Service List")
    Checkbutton(dictionary_window, text="NGSL", variable=var2b,
command=update_ngsl).grid(row=3, sticky=W)
    var3 = IntVar()
    def update_nltk_names():
        global names
        global use_nltk_names
        global nltk_names
        nonlocal var3
        if var3.get() == 1:
            use_nltk_names = True
            print(var3.get())
            print("loading NLTK personal names")
            nltk_names = names.words('male.txt') + names.words('female.txt')
            length = len(nltk_names)
            print("NLTK has ", length, " personal names.")
        else:
            use_nltk_names = False
            print(var3.get())
            print("not using NLTK personal names")
    Checkbutton(dictionary_window, text="NLTK personal names", variable=var3,
command=update_nltk_names).grid(row=4, sticky=W)
    var4 = IntVar()
    def update_wordnet():
        global use_wordnet
        nonlocal var4
        if var4.get() == 1:
            use_wordnet = True
            print(var4.get())
            print("using WordNet")
        else:

            use_wordnet = False
            print(var4.get())
            print("not using WordNet")
    Checkbutton(dictionary_window, text="Entire WordNet", variable=var4,
command=update_wordnet).grid(row=5, sticky=W)
    var5 = StringVar()
    subject_label = Entry(dictionary_window, text="Enter WordNet subject", textvariable =
var5, width = 30)
    subject_label.delete(0, END)
    subject_label.insert(0, "Enter WordNet subject")
    subject_label.grid(row=6, column=0)
    #e1 = Entry(dictionary_window)
    #e1.grid(row=5, column=1)
    def get_wn_subject_list():
        global use_wordnet_subject
        global wn_subject_words
        global wn_subjects
        wn_subject_synset = None
        for lemma in wn_subjects:
            if wn_subjects[lemma].get() == 1:
                wn_subject_synset = subject.synset
                print("Collecting wn for synset: ", synset.dictionary())
        hyponyms = set()
        for hyponym in wn_subject_synset.hyponyms():
            hyponyms |= set(get_hypo_meronyms(hyponym))
        part_meronyms = set()
        for meronym in wn_subject_synset.part_meronyms():
            part_meronyms |= set(get_hypo_meronyms(meronym))
        instance_hyponyms = set()
        for instance in wn_subject_synset.instance_hyponyms():
            instance_hyponyms |= set(get_hypo_meronyms(instance))
        wn_subject_words = \
            hyponyms | set(wn_subject_synset.hyponyms()) | \
            part_meronyms | set(wn_subject_synset.part_meronyms()) | \
            instance_hyponyms | set(wn_subject_synset.instance_hyponyms())
    var_wn_synset_id = IntVar()
    def wordnet_subject_update():
        global use_wordnet_subject
        global wn_subjects
        global wn_subject
        global wn_subject_words
        #nonlocal var5
        var_wn_synset_id = None
        wn_subject = var5.get()
        print("subject: ", wn_subject, " :")
        wn_synsets = wn.synsets(wn_subject)
        print("synsets: ", wn_synsets)
        wn_subject_words = []
        wn_subjects = {}
        wn_subject_window = Toplevel(main_window)
        wn_subject_window_title = 'Which sense of ' + wn_subject
        wn_subject_window.title(wn_subject_window_title)
        wn_subject_window.geometry('462x165+400+150')
        wn_subject_window.bg='brown'  # background color of edit area
        row_id = 1
        for synset in wn_synsets:
            wn_subjects[synset] = StringVar()
            definition = synset.definition()
            print("definition: ", definition)
            chk_btn = Checkbutton(wn_subject_window, text=definition,
variable=wn_subjects[synset], command=get_wn_subject_list)
            chk_btn.grid(row=row_id, sticky=W)
            row_id += 1
        dictionary_window.transient(main_window)
    Button(dictionary_window,
           text='Update WordNet Subject',
           command=wordnet_subject_update).grid(row=7, sticky=W, pady=4)
    var6 = IntVar()
    def update_foldoc():
        global use_foldoc
        nonlocal var6
        if var6.get() == 1:
            use_foldoc = True
            print(var6.get())
            print("using FOLDOC")
        else:
            use_foldoc = False
            print(var4.get())
            print("not using FOLDOC")
    Checkbutton(dictionary_window,
                text="FOLDOC 750,000 computer terms",
                variable=var6,
                command=update_foldoc).grid(row=8, sticky=W)
    #Button(dictionary_window, text='Show', command=show_dictionary).grid(row=7,
sticky=W, pady=4)
    def close_dictionary():
        dictionary_window.destroy()
    dictionary_window.protocol('WM_DELETE_WINDOW', close_dictionary) # override close
button
```

That new function produces this menu.



- [✓] Ogden Basic English 850 words
- [ ] VOA Simple English
- [ ] NGSL
- [ ] NLTK personal names
- [ ] Entire WordNet

Enter WordNet subject

- [ ] FOLDOC 750,000 computer terms

Function check_words then compares each word in the edit space with the selected sources. The text widget's get command returns the text from the edit space.

```
text = edit_space.get(0.1, END)
```

Next the natural language tool kit's word tokenizer function returns that text as a list of words.

```
tokens = word_tokenize(text)
```

Then each word, called token, is compared with the selected dictionaries. Notice FOLDOC is not downloaded because it is too large. Instead the program tries to open the Web page for that entry. If it succeeds then the word is good. That code is shown here.

```
if use_foldoc == True and acceptable == False:
    word_url = 'https://foldoc.org/' + token
    print("word_url: ", word_url)
    req = Request(word_url)
    try:
        response = urlopen(req)
    except URLError as e:
        if hasattr(e, 'reason'):
            print('We failed to reach FOLDOC server.')
            print('Reason: ', e.reason)
        elif hasattr(e, 'code'):
            print('The FOLDOC server couldn\'t fulfill the request.')
            print('Error code: ', e.code)
    else:
        # everything is fine
        raw_text = response.read()
        mystr = raw_text.decode("utf8")
        #print(raw_text)
        if "No match" not in mystr:
            acceptable = True;
```

The entire check_words function is displayed below.

```
def check_words():
    global edit_space
    global use_basic_english
    global basic_english_words
    global use_nltk_names
    global nltk_names
    global use_voa_simple_english
    global voa_simple_english_words
    global use_ngsl
    global ngsl_words
    global use_foldoc
    text = edit_space.get(0.1, END)
    #print(words)
    tokens = word_tokenize(text)
    punctuation_marks = [',', '.', ';', '?', ':']
    for token in tokens:
        if not(token in punctuation_marks):
            token_lower = token.lower()
            acceptable = False
            if use_basic_english == True:
                if token_lower in basic_english_words:
                    acceptable = True
            if use_voa_simple_english == True and acceptable == False:
                if token_lower in voa_simple_english_words:
                    acceptable = True
            if use_nltk_names == True and acceptable == False:
                if token in nltk_names:
                    acceptable = True
            if use_ngsl == True and acceptable == False:
                if token in ngsl_words:
                    acceptable = True
            if use_foldoc == True and acceptable == False:
                word_url = 'https://foldoc.org/' + token
                print("word_url: ", word_url)
                req = Request(word_url)
                try:
                    response = urlopen(req)
                except URLError as e:
                    if hasattr(e, 'reason'):
                        print('We failed to reach FOLDOC server.')
                        print('Reason: ', e.reason)
                    elif hasattr(e, 'code'):
                        print('The FOLDOC server couldn\'t fulfill the request.')
                        print('Error code: ', e.code)
                else:
                    # everything is fine
                    raw_text = response.read()
                    mystr = raw_text.decode("utf8")
                    #print(raw_text)
                    if "No match" not in mystr:
                        acceptable = True;
        if acceptable == False:
            tkinter.messagebox.showinfo(token," not in any of the chosen dictionaries")
```

And, the entire program is shown below.

```
import tkinter as tk
import tkinter.scrolledtext as tkst
import os
from tkinter.scrolledtext import ScrolledText
from tkinter import *
from tkinter import Menu
import tkinter.messagebox
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import words
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
from nltk.corpus import names
import urllib
from urllib.request import urlopen
from urllib.request import Request, urlopen
from urllib.error import  URLError
from nltk.corpus import wordnet as wn

# Dictionaries selected
basic_english_words = words.words('en-basic')
voa_simple_english_words = None
nltk_names = None

use_basic_english = True
use_voa_simple_english = False
use_nltk_names = False
use_csv_dictionary = False
```

```
use_ngsl = False
ngsl_words = None

use_wordnet = False
use_wordnet_subject = None
wn_subject = None
wn_subject_words = None

use_foldoc = False

main_window = tk.Tk(className="Special English Text Editor")
main_window.title('Special English Text Editor')
main_window.geometry('300x200')
edit_space = tkst.ScrolledText(
    master = main_window,
    wrap = 'word', # wrap text at full words only
    undo = TRUE,
    #width = 25, # characters
    #height = 10, # text lines
    bg='beige'  # background color of edit area
)

file_name = 'placeholder.txt'
dirty_edit_space = False

#def dummy():
    #print ("dummy Command to be removed later")

def undo(event = None):
    edit_space.event_generate("<<Undo>>") # <<Undo>> is TCL command, not python

def cut(event = None):
    edit_space.event_generate("<<Cut>>")  # <<Cut>> is TCL command

def copy(event = None):
    edit_space.event_generate("<<Copy>>") # <<Copy>> is TCL command

def paste(event = None):
    edit_space.event_generate("<<Paste>>") # <<Paste>> is TCL command

def on_key_press():
    global dirty_bit
    dirty_edit_space = True

# file icons
open_file_icon = PhotoImage(file='icons/open_file.gif')
save_file_icon = PhotoImage(file='icons/Save.gif')
```

```
# edit icons
undo_icon = PhotoImage(file='icons/Undo.gif')
cut_icon = PhotoImage(file='icons/Cut.gif')
copy_icon = PhotoImage(file='icons/Copy.gif')
pasteicon = PhotoImage(file='icons/Paste.gif')

def open_file(event=None):
    global file_name
    dirty_edit_space = False
    file_name = filedialog.askopenfilename(defaultextension=".txt",filetypes=[("All
Files","*.*"),("Text Documents","*.txt")])
    if file_name == "": # If no file chosen.
        file_name = None # Absence of file.
    else:
        main_window.title(os.path.basename(file_name)) # Returning the basename of 'file'
        edit_space.delete(1.0,END)
        fh = open(file_name,"r")
        edit_space.insert(1.0,fh.read())
        fh.close()

# save file function
def save_file():
    global file_name
    global dirty_edit_space
    try:
        f = open(file_name, 'w')
        content = edit_space.get(1.0, 'end')
        f.write(content)
        f.close()
        dirty_edit_space = False
    except:
        file_name = None

def save_as():
    global file_name
    global dirty_edit_space
    file_name = filedialog.asksaveasfilename()
    try:
        f = open(file_name, 'w')
        content = edit_space.get(1.0, 'end')
        f.write(content)
        f.close()
        dirty_edit_space = False
    except:
        file_name = None

def exit_main_window():
    global dirty_edit_space
    if dirty_edit_space:
        isOK = tkinter.messagebox.askokcancel("Go ahead w/o saving new edits?", "OK?")
        if isOK:
          main_window.destroy
          dirty_edit_space = False

def on_find():
    find_window = Toplevel(main_window)
    find_window.title('Find')
    find_window.geometry('262x65+200+250')
    find_window.transient(main_window)
    find_window.bg='beige'  # background color of edit area
    Label(find_window, text="Find All:").grid(row=0, column=0, sticky='e')
    find_value = StringVar()
    entry_field = Entry(find_window, width=25, textvariable = find_value)
    entry_field.grid(row=0, column=1, padx=2, pady=2, sticky='we')
    entry_field.focus_set()
    ignore_case = IntVar()
    Checkbutton(find_window, text='Ignore Case', variable=ignore_case).grid(row=1,
column=1, sticky='e', padx=2, pady=2)
    Button(
        find_window,
        text = "Find All",
        underline = 0,
        command = lambda: search_for(find_value.get(), ignore_case.get(), edit_space,
find_window, entry_field)).grid(row = 0, column = 2, sticky = 'e' + 'w', padx = 2, pady = 2)
    def close_search():
        edit_space.tag_remove('match', '1.0', END)
        find_window.destroy()
    find_window.protocol('WM_DELETE_WINDOW', close_search) # override close button

# find the needle in the haystack
def search_for(needle, ignore_case, text_haystack, find_window, entry_field):
    edit_space.tag_remove('match', '1.0', END)
    count =0
    if needle:
        pos = '1.0'
        while True:
            pos = text_haystack.search(needle, pos, nocase = ignore_case, stopindex = END)
            if not pos: break
            lastpos = '%s+%dc' % (pos, len(needle))
            text_haystack.tag_add('match', pos, lastpos)
            count += 1
            pos = lastpos
        text_haystack.tag_config('match', foreground='red', background='yellow')
    entry_field.focus_set()
    find_window.title('%d matches found' %count)

#
# Dictionaries selected
```

```
#basic_english_words = words.words('en-basic')
#use_basic_english = True
#use_voa_simple_english = False
#use_csv_dictionary = False
#use_wordnet = False
#use_wordnet_subject = None
#
def check_words():
    global edit_space
    global use_basic_english
    global basic_english_words
    global use_nltk_names
    global nltk_names
    global use_voa_simple_english
    global voa_simple_english_words
    global use_ngsl
    global ngsl_words
    global use_foldoc
    text = edit_space.get(0.1, END)
    #print(words)
    tokens = word_tokenize(text)
    punctuation_marks = [',', '.', ';', '?', ':']
    for token in tokens:
        if not(token in punctuation_marks):
            token_lower = token.lower()
            acceptable = False
            if use_basic_english == True:
                if token_lower in basic_english_words:
                    acceptable = True
            if use_voa_simple_english == True and acceptable == False:
                if token_lower in voa_simple_english_words:
                    acceptable = True
            if use_nltk_names == True and acceptable == False:
                if token in nltk_names:
                    acceptable = True
            if use_ngsl == True and acceptable == False:
                if token in ngsl_words:
                    acceptable = True
            if use_foldoc == True and acceptable == False:
                word_url = 'https://foldoc.org/' + token
                print("word_url: ", word_url)
                req = Request(word_url)
                try:
                    response = urlopen(req)
                except URLError as e:
                    if hasattr(e, 'reason'):
                        print('We failed to reach FOLDOC server.')
                        print('Reason: ', e.reason)
                    elif hasattr(e, 'code'):
```

```
                    print('The FOLDOC server couldn\'t fulfill the request.')
                    print('Error code: ', e.code)
                else:
                    # everything is fine
                    raw_text = response.read()
                    mystr = raw_text.decode("utf8")
                    #print(raw_text)
                    if "No match" not in mystr:
                        acceptable = True;
        if acceptable == False:
            tkinter.messagebox.showinfo(token," not in any of the chosen dictionaries")

def select_dictionaries():
    global use_basic_english
    global use_voa_simple_english
    global voa_simple_english_words
    global names
    global use_nltk_names
    global nltk_names
    global use_wordnet
    global use_wordnet_subject
    global wn_subject
    global use_csv_dictionary
    global use_foldoc
    #global var1
    #global var2
    #global var3
    #global var4
    wn_subjects = {}
    dictionary_window = Toplevel(main_window)
    dictionary_window.title('Dictionaries')
    dictionary_window.geometry('262x365+200+250')
    dictionary_window.bg='beige'  # background color of edit area
    dictionary_window.transient(main_window)
    Label(dictionary_window, text="Check all appropriate sources:").grid(row=0, column=0,
sticky='e')
    var1 = IntVar()
    def update_basic_english():
        global use_basic_english
        nonlocal var1
        if var1.get() == 1:
            use_basic_english = True
            print(var1.get())
            print("using Basic English")
        else:
            use_basic_english = False
            print(var1.get())
            print("not using Basic English")
    chk_btn_basic = Checkbutton(dictionary_window, text="Ogden Basic English 850 words",
variable=var1, command=update_basic_english)
    chk_btn_basic.grid(row=1, sticky=W)
    chk_btn_basic.select()
    var2 = IntVar()
    def update_voa_simple_english():
        global use_voa_simple_english
        global voa_simple_english_words
        nonlocal var2
        if var2.get() == 1:
            use_voa_simple_english = True
            print(var2.get())
            print("loading VOA Simple English")
            voa_simple_english_words_corpus = PlaintextCorpusReader('./','voaSpecialEnglish.txt')
            file_ids = voa_simple_english_words_corpus.fileids()
            #print("VOA Simple English file info")
            #for f in file_ids:
                #print("file id: ", f)
            voa_simple_english_words = voa_simple_english_words_corpus.words() # returns list
of string
            length = len(voa_simple_english_words)
            print("VOA Special English has ", length, " words.")
        else:
            use_voa_simple_english = False
            print(var2.get())
            print("not using VOA Simple English")
    Checkbutton(dictionary_window, text="VOA Simple English", variable=var2,
command=update_voa_simple_english).grid(row=2, sticky=W)
    var2b = IntVar()
    def update_ngsl():
        global use_ngsl
        global ngsl_words
        nonlocal var2b
        if var2b.get() == 1:
            use_ngsl = True
            print(var2b.get())
            print("loading New General Service List")
            ngsl_words_corpus = PlaintextCorpusReader('./','ngsl_Headwords.txt')
            #file_ids = voa_simple_english_words_corpus.fileids()
            #print("VOA Simple English file info")
            #for f in file_ids:
                #print("file id: ", f)
            ngsl_words = ngsl_words_corpus.words() # returns list of string
            length = len(ngsl_words)
            print("NGSL New General Service List has ", length, " words.")
        else:
            use_ngsl = False
            print(var2b.get())
            print("not using NGSL New General Service List")
    Checkbutton(dictionary_window, text="NGSL", variable=var2b,
command=update_ngsl).grid(row=3, sticky=W)
    var3 = IntVar()
    def update_nltk_names():
        global names
        global use_nltk_names
        global nltk_names
        nonlocal var3
        if var3.get() == 1:
            use_nltk_names = True
            print(var3.get())
            print("loading NLTK personal names")
            nltk_names = names.words('male.txt') + names.words('female.txt')
            length = len(nltk_names)
            print("NLTK has ", length, " personal names.")
        else:
            use_nltk_names = False
            print(var3.get())
            print("not using NLTK personal names")
    Checkbutton(dictionary_window, text="NLTK personal names", variable=var3,
command=update_nltk_names).grid(row=4, sticky=W)
    var4 = IntVar()
    def update_wordnet():
        global use_wordnet
        nonlocal var4
        if var4.get() == 1:
            use_wordnet = True
            print(var4.get())
            print("using WordNet")
        else:

            use_wordnet = False
            print(var4.get())
            print("not using WordNet")
    Checkbutton(dictionary_window, text="Entire WordNet", variable=var4,
command=update_wordnet).grid(row=5, sticky=W)
    var5 = StringVar()
    subject_label = Entry(dictionary_window, text="Enter WordNet subject", textvariable =
var5, width = 30)
    subject_label.delete(0, END)
    subject_label.insert(0, "Enter WordNet subject")
    subject_label.grid(row=6, column=0)
    #e1 = Entry(dictionary_window)
    #e1.grid(row=5, column=1)
    def get_wn_subject_list():
        global use_wordnet_subject
        global wn_subject_words
        global wn_subjects
        wn_subject_synset = None
        for lemma in wn_subjects:
            if wn_subjects[lemma].get() == 1:
                wn_subject_synset = subject.synset
                print("Collecting wn for synset: ", synset.dictionary())
        hyponyms = set()
        for hyponym in wn_subject_synset.hyponyms():
            hyponyms |= set(get_hypo_meronyms(hyponym))
        part_meronyms = set()
        for meronym in wn_subject_synset.part_meronyms():
            part_meronyms |= set(get_hypo_meronyms(meronym))
        instance_hyponyms = set()
        for instance in wn_subject_synset.instance_hyponyms():
            instance_hyponyms |= set(get_hypo_meronyms(instance))
        wn_subject_words = \
            hyponyms | set(wn_subject_synset.hyponyms()) | \
            part_meronyms | set(wn_subject_synset.part_meronyms()) | \
            instance_hyponyms | set(wn_subject_synset.instance_hyponyms())
    var_wn_synset_id = IntVar()
    def wordnet_subject_update():
        global use_wordnet_subject
        global wn_subjects
        global wn_subject
        global wn_subject_words
        #nonlocal var5
        var_wn_synset_id = None
        wn_subject = var5.get()
        print("subject: ", wn_subject, " :")
        wn_synsets = wn.synsets(wn_subject)
        print("synsets: ", wn_synsets)
        wn_subject_words = []
        wn_subjects = {}
        wn_subject_window = Toplevel(main_window)
        wn_subject_window_title = 'Which sense of ' + wn_subject
        wn_subject_window.title(wn_subject_window_title)
        wn_subject_window.geometry('462x165+400+150')
        wn_subject_window.bg='brown'  # background color of edit area
        row_id = 1
        for synset in wn_synsets:
            wn_subjects[synset] = StringVar()
            definition = synset.definition()
            print("definition: ", definition)
            chk_btn = Checkbutton(wn_subject_window, text=definition,
variable=wn_subjects[synset], command=get_wn_subject_list)
            chk_btn.grid(row=row_id, sticky=W)
            row_id += 1
        dictionary_window.transient(main_window)
    Button(dictionary_window,
           text='Update WordNet Subject',
           command=wordnet_subject_update).grid(row=7, sticky=W, pady=4)
    var6 = IntVar()
    def update_foldoc():
        global use_foldoc
        nonlocal var6
        if var6.get() == 1:
            use_foldoc = True
            print(var6.get())
            print("using FOLDOC")
        else:
            use_foldoc = False
            print(var4.get())
            print("not using FOLDOC")
    Checkbutton(dictionary_window,
                text="FOLDOC 750,000 computer terms",
                variable=var6,
                command=update_foldoc).grid(row=8, sticky=W)
    #Button(dictionary_window, text='Show', command=show_dictionary).grid(row=7,
sticky=W, pady=4)
    def close_dictionary():
        dictionary_window.destroy()
    dictionary_window.protocol('WM_DELETE_WINDOW', close_dictionary) # override close
button


def about_desc():
    tkinter.messagebox.showinfo("About Special English Text Editor",
        "An open source python-based program to encourage writing text in Special English"
        )

main_menu = Menu(main_window)
main_window.config(menu=main_menu)

file_menu = Menu(main_menu)
main_menu.add_cascade(
    label="File",
    menu=file_menu)
file_menu.add_command(
    label="Open",
    accelerator='Ctrl+O',
    compound=LEFT,
    image=open_file_icon,
    underline = 0,
    command=open_file
    )

file_menu.add_command(
    label="Save",
    accelerator='Ctrl+S',
    compound=LEFT,
    image=save_file_icon,
    underline=0,
    command=save_file
    )
file_menu.add_command(label="Save As", command = save_as)
file_menu.add_separator()
file_menu.add_command(
    label="Exit",
    command = exit_main_window
    )

edit_menu = Menu(main_menu)
main_menu.add_cascade(label="Edit", menu=edit_menu)
edit_menu.add_command(
    label="Undo",
    compound=LEFT,
    image=undo_icon,
    accelerator='Ctrl+Z',
    command = undo)
edit_menu.add_separator()
edit_menu.add_command(
    label="Cut",
    compound = LEFT,
    image = cut_icon,
    accelerator='Ctrl+X',
    command = cut)
edit_menu.add_command(
    label = "Copy",
    compound = LEFT,
    image = copy_icon,
    accelerator='Ctrl+C',
    command = copy)
edit_menu.add_command(
    label="Paste",
    compound=LEFT,
    image=pasteicon,
    accelerator='Ctrl+V',
    command = paste)
edit_menu.add_separator()
edit_menu.add_command(
    label = "Find",
    underline = 0,
    accelerator = 'Ctrl+F',
    command = on_find)
#edit_menu.add_command(label="Select All", underline=7, accelerator='Ctrl+A',
command=select_all)
edit_menu.add_command(
    label="Words",
    #underline= 0,
    #accelerator='Ctrl+W',
            command = check_words)
        edit_menu.add_command(
            label="Dictionary",
            #underline= 0,
            #accelerator='Ctrl+W',
            command = select_dictionaries)

        help_menu = Menu(main_menu)
        main_menu.add_cascade(label="Help", menu=help_menu)
        help_menu.add_command(label="About...", command=about_desc)
        # end of menu creation
        edit_space.pack(fill = BOTH, expand = 1)

        #edit_space.bind('<Control-N>', new_file)
        #edit_space.bind('<Control-n>', new_file)
        edit_space.bind('<Control-O>', open_file)
        edit_space.bind('<Control-o>', open_file)
        edit_space.bind('<Control-S>', save_file)
        edit_space.bind('<Control-s>', save_file)
        #edit_space.bind('<Control-A>', select_all)
        #edit_space.bind('<Control-a>', select_all)
        edit_space.bind('<Control-f>', on_find)
        edit_space.bind('<Control-F>', on_find)
        #edit_space.bind('<KeyPress-F1>', help_box)
        edit_space.bind('<KeyPress-F1>', on_key_press)

        main_window.mainloop()
```

Thus we have seen how easily the text widget supports complex projects. There is a lot of interesting code supporting FOLDOC and WordNet. Wikipedia has a description of FOLDOC at https://en.wikipedia.org/wiki/Free_On-line_Dictionary_of_Computing and WordNet at https://en.wikipedia.org/wiki/WordNet. Each dictionary would require extensive effort to understand both its organization and to write python code to employ that dictionary.

The text widget is highly regarded for its functionality. It distinguishes Tcl/Tk from all other python GUI packages.